Kari Niskanen

JÄTKÄNSHAKKIPELIAUTOMAATTI

Insinöörityö
Kajaanin ammattikorkeakoulu
Tekniikan ja liikenteen ala
Tietotekniikan koulutusohjelma
2006

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Tekniikka	Koulutusohjelma Tietotekniikka
Tekijä(t) Kari Niskanen	
Työn nimi Jätkänshakkipeliautomaatti	
Vaihtoehtoiset ammattiopinnot Ohjelmistotekniikka	Ohjaaja(t) Risto Airaksinen
	Toimeksiantaja Kajaanin ammattikorkeakoulu
Aika 16.11.2006	Sivumäärä ja liitteet 38

Insinöörityön tarkoituksena oli suunnitella ja toteuttaa Kajaanin ammattikorkeakoululle jätkänshakkipeliautomaatin ohjelmakoodi. Jätkänshakkipeliautomaattia on tarkoitus käyttää eri tilaisuuksissa, kuten esimerkiksi messuilla, joissa esitellään Kajaanin ammattikorkeakoulua sekä sen eri koulutusaloja. Peliautomaatin tarkoituksena on esitellä tietotekniikankoulutusohjelman tarjoamia mahdollisuuksia.

Ohjelma toteutettiin c-kielellä 8051-mikrokontrollerille. Laitteistona peliautomaatissa käytettiin valmista testilaitetta, joka sisältää graafisen LCD-näytön, 8051:n ja matriisinäppäimistön. Kehitysympäristönä työssä käytettiin IAR ANSI C -kääntäjää sekä Intel 8051:n emulaattoria. Peliä pelataan tietokonepelaajaa vastaan. Tietokonepelaajan tekoälyn toteutuksen tavoitteena oli toteuttaa ns. itseoppivatekoäly, eli ensimmäisessä pelissä tietokone tietää ainoastaan säännöt, jonka jälkeen pelien edetessä tietokone oppii pelin eri strategiat. Kun ns. oppiminen on tapahtunut, lopputuloksena tulee aina tasapeli tai tietokone voittaa.

Ohjelmaa testattiin eri ohjelmointivaiheissa. Testauksessa saatujen testitulosten perusteella voitiin havaita, että kyseinen ohjelma toimi odotusten mukaisesti. Lopputuloksena saatiin ohjelma, joka sisälsi laitteiston tarvitsemat ohjelmat ja jätkänshakkipelin pääohjelman varustettuna tietokonepelaajan tekoälyllä.

Kieli	Suomi
Asiasanat	Tekoäly, 8051, ristinolla
Säilytyspaikka	☐ Kajaanin ammattikorkeakoulun Kaktus-tietokanta ☐ Kajaanin ammattikorkeakoulun kirjasto

Kajaanin ammattikorkeakoulu
University of Applied Sciences

THESIS
ABSTRACT

School School of Engineering	Degree Programme Information Technology
Author(s) Kari Niskanen	
Title The Tic-Tac-Toe Game Machine	
Optional Professional Studies Software Technology	Instructor(s) Risto Airaksinen
	Commissioned by Kajaani University of Applied Sciences
Date November 16, 2006	Total Number of Pages and Appendices 38

The purpose of this Bachelor's thesis was to design and implement a code of the tic-tac-toe game machine for Kajaani University of Applied Sciences. The Tic-tac-toe game machine will be used in different gatherings, such as exhibitions, in which Kajaani University of Applied Sciences and its degree programs are introduced. The game machine will be used to demonstrate the opportunities of the Information Technology degree program.

The software was implemented with c-code to the Intel 8051 microcontroller. The existing embedded hardware, including an LCD display, the 8051 microcontroller and a matrix keyboard, was used as the hardware platform. The IAR ANSI C complier and the Intel 8051 emulator used as a software development environment. The game is played against a computer player. The target was to develop the computer player "selfeducated" artificial intelligence. This means that in the first game the computer understands only the rules of the game and as the game proceeds the computer player learns different game strategies, tricks etcetera. When learning has occurred, the game always ends either to a draw or the computer wins.

The software was tested during the different development phases. The testing results show that the software functions as expected. As the final result, well working software was achieved including the software required by the embedded HW platform as well as the game main software program with artificial intelligence.

Language of Thesis	Finnish
Keywords	Artificial intelligence, 8051, tic-tac-toe
Deposited at	☐ Kaktus Database at Kajaani University of Applied Sciences ☐ Library of Kajaani University of Applied Sciences

Lyhenteet, käsitteet ja määritelmät

AI	Artificial Intelligence, tekoäly
CPU	Central Processing Unit, Suoritin eli prosessori on tietokoneen sydän, joka suorittaa tietokoneohjelman sisältämiä konekielisiä käskyjä.
De Facto	De facto on latinankielinen käsite, joka tarkoittaa "käytännössä" tai "yleisesti", yleensä de juren ("lain mukaan") vastakohtana. Asiat, jotka ovat de facto, yleensä kehittyvät autonomisesti, ilman erillistä standardin, lain, säännön tai sopimuksen tekoa.
ROM	ROM-muisti (Read-Only Memory), lukumuisti on tietokoneen pysyväismuisti, johon ei voi tehdä muutoksia normaalikäytön aikana ja jonka tiedot säilyvät silloinkin, kun tietokoneesta kytketään virta pois.
UML	Unified Modelling Language, UML-mallinnus. UML-mallinnus (Unified Modelling Language) on Object Management Groupin (OMG) vuonna 1997 standardoima graafinen mallinnuskieli, joka sisältää 13 erilaista kaaviota. Kaavioista kuudella kuvataan rakennetta, kolmella käyttäytymistä ja neljällä vuorovaikutusta. UML on alun perin kehitetty järjestelmä- ja ohjelmistokehitystä varten.
VRAM	Video RAM, näyttömuisti. Kaksiporttinen DRAM-piiri, voidaan lukea ja kirjoittaa samanaikaisesti, käytetään yleensä näytönohjaimissa.

SISÄLLYS

1 JOHDANTO

Insinöörityön tarkoituksena oli suunnitella ja toteuttaa jätkänshakkipeliautomaatin ohjelmakoodi. Peliautomaatti tulee Kajaanin ammattikorkeakoulun käyttöön, joka on myös työn tilaaja. Jätkänshakkipeliautomaattia on tarkoitus käyttää eri tilaisuuksissa, kuten esimerkiksi messuilla, joissa esitellään Kajaanin ammattikorkeakoulua sekä sen eri koulutusaloja. Peliautomaatin tarkoituksena on markkinoida tietotekniikan koulutusohjelman tarjoamia mahdollisuuksia.

Laitteistona peliautomaatissa käytetään valmista testilaitetta, joka sisältää graafisen LCD-näytön, 8051-mikrokontrollerin ja matriisinäppäimistön.

Jätkänshakki on peli, jossa on 3 x 3 ruudukko ja pelimerkkeinä on O tai X. Pelin tarkoituksena on saada kolmen merkin pysty-, vino- tai vaakasuorarivi aikaiseksi. Peliä pelataan tietokonetta vastaan. Pelin tuli olla ns. itseoppiva, eli ensimmäisessä pelissä tietokone tietää ainoastaan säännöt, jonka jälkeen pelien edetessä tietokone oppii pelin eri strategiat. Kun ns. oppiminen on tapahtunut, lopputuloksena tulee aina tasapeli tai tietokone voittaa.

Insinöörityön aluksi käydään läpi hieman jätkänshakkipelin sääntöjä ja sen syntyhistoriaa. Tämän jälkeen luodaan katsaus peliteoriaan työn aiheen tiimoilta. Sen jälkeen esitellään työssä käytetyn laitteiston pääosat. Lopuksi kuvataan työn toteutus- ja testausosiot.

2 JÄTKÄNSHAKKIPELIN SÄÄNNÖT JA HISTORIA

Jätkänshakkipeli on yksi variaatio ristinollapelistä, joka tunnetaan maailmalla nimellä TIC-TAC-TOE. Jätkänshakki on kahden pelaajan peli, jossa on 3 x 3 peliruudukko. Pelaajat piirtävät vuorotellen ruudukkoon omat merkkinsä, jotka ovat O tai X. Pelin tavoitteena on saada aikaan peliruudukolle kolmen merkin suora, joko pystyyn, vaakaan tai vinoon. Pelissä päädytään aina tasapeliin, jos oletetaan, että kummatkin pelaajat pelaavat parhaalla mahdollisella tavalla eivätkä tee virheitä. Laajemmassa versiossa peliruudukko on periaatteessa ääretön, jossa on tarkoituksena saada viiden tai kuuden merkin suora joko vaakaan, pystyyn tai vinoon.

Ristinolla pohjautuu alun perin peliin nimeltä Gomoku. Gomoku on lautapeli, jonka juuret ulottuvat muinaiseen Kiinaan ja Japaniin. Pelissä pelataan 15 vaaka- ja 15 pystyviivan 225 leikkauspisteestä koostuvalla pelilaudalla. Kaksi pelaajaa, musta ja valkoinen pelaavat vuorotellen asettaen omaa väriä olevan kiven tyhjälle leikkauspisteelle. Musta aloittaa pelin. Pelaaja, joka ensimmäisenä saa aikaan viiden kiven suoran (pystysuoraan, vaakasuoraan tai viistoon), voittaa pelin. Kerran pelilaudalle asetettua kiveä ei voida liikuttaa eikä poistaa. Jos pelilauta tulee täyteen eikä viiden suoraa ole saatu aikaan, tuloksena on tasapeli. "Standardi-Gomokussa" suoralla, jolla on useampia kuin viisi kiveä (esim. kuusi), ei voita.[1.]

3 PELITEORIAA

Peliteoria on syntynyt 1900-luvulla. Se on jatkuvasti laajentuva tieteenala. Sen voi käsittää pelaajan oppaana ja eri pelien analysointityökaluna. Peli taas voi olla melkein mitä tahansa, kuten esimerkiksi shakki, tietokonepeli tai vaikkapa New Yorkin pörssi. Peliteoria onkin siis sovellusalueena hyvinkin laaja.[2.]

Peliteoria ja AI (Artificial Intelligence) eli tekoäly ovat jokseenkin samassa suhteessa kuin matematiikka ja tietojenkäsittelytiede. AI sisältää epävarmuutta, kuten tietojenkäsittelytiede yleensä. Peliteoria on taas täysin determinististä ja todistettavasti varmaa kuten matematiikka.[2.]

Peliteoriaa voidaan soveltaa tilanteisiin, joissa päätöksentekijä joutuu ottamaan huomioon muitten päätöksentekijöiden valinnat. Kun otetaan huomioon strategiset näkökulmat eli asiat joita kontrolloivat päätöksentekijät eikä puhdas sattuma, saadaan aikaan teorioita, jotka sekä täydentävät että menevät klassista todennäköisyysteoriaa pidemmälle. Peliteoriaa on käytetty esimerkiksi päättämässä poliittisten ja taloudellisten koalitioiden rakennetta, tuotteiden ja palveluiden optimaalista hintaa, äänestäjäjoukon vaikutusvaltaa vaaleissa, valamiehistön valtaa oikeudessa, parasta paikkaa tehtaalla ja jopa tiettyjen elämästä taistelevien lajien käyttäytymistä. Olisikin aika hämmästyttävää, jos mikään muu tieteenala kattaisi yhtä laajaa sovellusaluetta. Ei ole olemassa mitään tiettyä peliteoriaa, jota voitaisiin käyttää aina, vaan on olemassa paljon erilaisia teorioita, joita sovelletaan erilaisiin tilanteisiin.[2.]

3.1 Klassinen peliteoria

Klassinen peliteoria tutkii erilaisten strategioiden hyvyyttä yksinkertaisissa lineaarisissa peleissä, kuten esimerkiksi "kivi, sakset, paperi" -pelissä. Lineaarisilla peleillä (Linear Game) tarkoitetaan pelejä, jotka päättyvät tehtyjen siirtojen suhteen lineaarisessa ajassa.

Seuraavaksi tarkastellaankin "kivi, sakset, paperi" -pelin muutamaa strategiaa. Valitaan strategiaksi vaikka sellainen, että ”olen aina kivi” tai ”olen aina paperi todennäköisyydellä A ja sakset todennäköisyydellä B jne. Tarkoituksena on siis löytää strategia, jolla peli voittaa suurimmalla mahdollisella todennäköisyydellä, joka vaihtelee välillä 0…1.[2.]

3.2 Peliteoreettiset lautapelit

Peliteoreettiselle tarkastelulle soveltuvat erityisesti hyvin lautapelit, koska niiden analysointi on yleensä helppoa. Perinteisen peliteorian ”kuninkuuspeli” oli shakki, kunnes Deep Blue voitti hallitsevan maailmanmestarin Garri Kasparovin vuonna 1997. Tämän jälkeen suurin mielenkiinto ja suurimmat haasteet ovat päätyneet Go-pelin ylle, vaikkakin pelin matemaattisen analyysin katsotaan alkaneen jo vuonna 1989. Siltikin Go:n etsintäavaruus on niin suuri, että hyvien heuristiikkojen löytäminen on erittäin hankalaa. Heuristiikalla tarkoitetaan löytämisen ja keksimisen tiedettä ja taidetta. Sana tulee kreikan kielen ilmaisusta heureka, "löysin". Parhaimmatkin Go-tietokoneohjelmat ovat nykyisin vain auttavaa amatööritasoa, ja on sanottu, että maapallolla on miljoona ihmistä, jotka voittavat parhaimmankin Go-ohjelman.[2.]

3.3 Kahden pelaajan täyden informaation nollasummapelit

Nollasummapeli on peliteoreettinen tilanne, jossa yhden toimijan voitot ja saavutukset ovat aina pois muilta toimijoilta, tai päinvastoin. Peliä kutsutaan nollasummapeliksi sen vuoksi, koska jos kaikkien pelaajien tappiot ja voitot lasketaan yhteen, tuloksena on aina nolla. Täydellä informaatiolla tarkoitetaan sitä, että tiedossa on kaikki pelin seuraavat mahdolliset siirrot ja niiden vaikutukset.[3.]

Suurin osa lautapeleistä on täyden informaation pelejä, koska niiden säännöt määräävät kaikki mahdolliset siirrot ja näiden aiheuttamat vaikutukset pelin kulkuun. Suurin osa perinteisistä kahden pelaajan lautapeleistä, kuten shakki, tammi, mylly ja Go, on kahden pelaajan täyden informaation nollasummapelejä. Näissä kaksi pelaajaa mittelee toisiaan vastaan tehden vuorotellen siirtoja tavoitellen vastakkaisia päämääriä.[3.]

Myös jätkänshakki lukeutuu kahden pelaajan täyden informaation nollasummapeleihin. Tässäkin pelissä toisen pelaajan ”hyvä” siirto on toisen pelaajan kannalta ”huono”. Pelissä on tietyt säännöt, pelissä tiedetään kaikki seuraavat siirrot ja niiden vaikutukset. Jätkänshakissa onkin mahdollista laskea lopputulos ensimmäisen siirron jälkeen, mikäli oletetaan, että kummatkin osapuolet pelaavat parhaalla mahdollisella tavalla. Tämän vuoksi jätkänshakki lukeutuu myös ns. ratkaistuihin peleihin. Taulukossa 1 on esitetty muutamia ratkaistuja ja osittain ratkaistuja pelejä.

Taulukko 1. Ratkaistuja ja osittain ratkaistuja pelejä.[4.]

Peli	Ratkaistu	Osittain ratkaistu	Lisätietoja
Awari	√		
Connect Four	√		
GoMoku	√		
Hex	√		9 × 9 -laudalla
Nim	√		
Mylly	√		
Jätkänshakki	√		Three Men’s Morris
Qubic	√		4 × 4 × 4 ristinolla
Ristinolla	√		
Tammi		√	
Shakki		√	
Go		√	4 × 4 -laudalla
Reversi		√	

4 KÄYTETTÄVÄ LAITTEISTO JA YMPÄRISTÖ

Ohjelma on toteutettu valmiiseen testilaitteistoon, joka sisältää seuraavat pääosat:

- Graafinen LCD-näyttö (Hitachin HD1830)
- Mikrokontrolleri 8051
- 4 x 4 matriisinäppäimistö

Kuvassa 1 on esitetty työssä käytetty testilaite.

Kuva 1. Työssä käytetty testilaite.

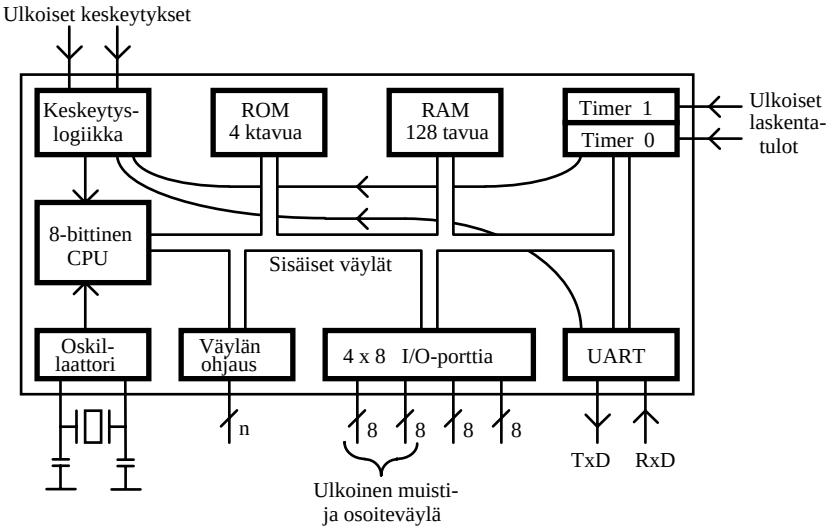
4.1 Mikrokontrolleri 8051

MCS (MicroController System)-51-piiriperhe on alun perin Intelin suunnittelema mikrokontrolleriperhe, johon kuuluu kaikkiaan useita kymmeniä eri piirejä. Toisistaan piirit poikkeavat piirille integroidun muistin määrän ja tyypin sekä piirille integroitujen I/O-ominaisuuksien puolesta. Kaikki piirit kumminkin omaavat yhteisen käskykannan. MC-51-sarjan prosessoreille on useita valmistajia, kuten esim. Intel, AMD, Philips ja Siemens. [5.]

MCS-51-piiriperheen peruspiiri 8051:n pääominaisuudet ovat seuraavat:

- 8-bittinen keskusyksikkö
- 64 kilotavun muistiavaruus ohjelmamuistille
- 64 kilotavun muistiavaruus datamuistille
- 4 kilotavua sisäistä (on-chip) ohjelmamuistia, ROM
- 128 tavua sisäistä datamuistia, RAM
- 32 kaksisuuntaista ja yksittäin osoitettavaa I/O-linjaa, jotka on ryhmitelty neljäksi 8 bitin portiksi P0, P1, P2 ja P3
- kaksi 16-bittistä ajastinta/laskuria
- täysin kaksisuuntainen sarjaportti (full duplex UART, Universal Asynchronous Receiver/Transmitter)
- vektoroitu keskeytysrakenne, jossa kaksi prioriteettitasoa
- sisäinen kello-oskillaattori
- laaja käskykanta: kuusi osoitusmuotoa, yksittäisten bittien osoitusmahdollisuus[5.]

Alla oleva kuva esittää 8051-mikrokontrollerin lohkotason sisäistä rakennetta[5.]



Kuva 2. 8051-mikrokontrollerin lohkotason sisäinen rakenne. [1.]

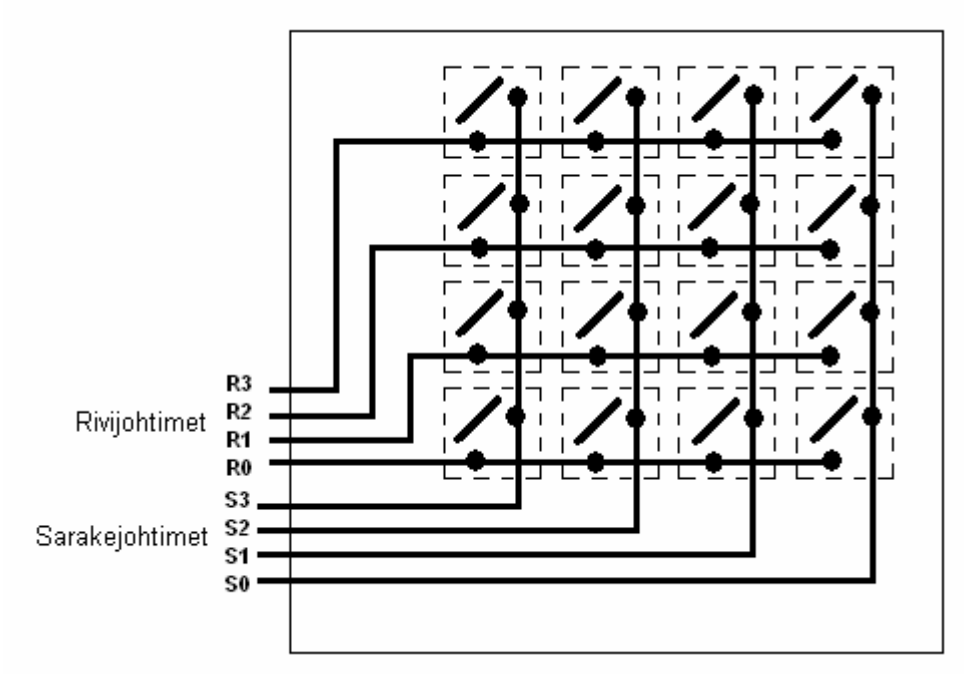
4.2 Graafinen LCD-näyttö

Laitteistossa käytettään Hitachin yksiväristä graafista LCD-näyttöä. Näyttö voidaan käyttää graafisessa tai tekstitilassa. Näytön ohjauksessa käytetään Hitachin HD1830-piiriä, joka on integroitu LCD-näyttöyksikköön. Piirillä on näyttömuisti (VRAM) ja sisäänrakennettu 192 merkin merkkigeneraattori ROM (CGROM). Piirillä on grafiikkatilassa maksimissaan 64 kt ulkoista näyttömuistia. Tekstitilassa ulkoista näyttömuistia on 4 kt. Kuvassa 3 on esitetty laitteistossa käytetty graafinen LCD-näyttö.

Kuva 3. Graafinen LCD-näyttö.

4.3 Matriisinäppäimistö

CPU:lle kytketty yksittäinen painonappi tai kytkin näkyy omana bittinä, joka voidaan lukea. Matriisinäppäimistöltä ei sen sijaan saadakaan jokaiselta näppäimeltä omaa erillistä signaalia, vaan se on toteutettu eri tavalla. Näppäimen painaminen vastaa sitä, että yksi rivi- ja yksi sarakejohdin menevät keskenään kontaktiin (ks. kuva 4). Kuvassa 4 on esitetty 4 x 4 matriisinäppäimistön kytkentä. [6.]



Kuva 4. Laitteistossa olevan 4 x 4 -matriisinäppäimistön sisäinen kytkentä [6.]

Laitteistossa käytetään matriisinäppäimistöä, joka sisältää seuraavat näppäimet: 1, 2, 3, 4, 5, 6, 7, 8, 9, F, E, D ja C. Kuvassa 5 on esitetty laitteistossa käytetty näppäimistö.

Kuva 5. Matriisinäppäimistö.

Näppäimistöllä ei ole valmiina käyttöjännitettä, joiden perusteella siltä saataisiin loogista ”0”- tai ”1”-tietoa. Matriisinäppäimistöltä ei voida suoraan yhdellä lukemisella saada tietoa painetusta näppäimestä. Painettu näppäin pitää selvittää sarjalla kirjoitus- ja lukujaksoja, joilla käydään läpi rivi kerrallaan (tai sarake kerrallaan) koko näppäimistö.[6.]

Näppäimistö pitää ”skannata” painetun näppäimen selville saamiseksi. Periaatteena on, että rivijohtimiin kirjoitetaan yhteen kerrallaan ”1”-tila, joka siis vastaa käyttöjännitteen syöttämistä vuorotellen eri riveihin. Sarakejohtimia lukemalla selvitetään, että minkä rivin ja sarakkeen risteyskohdassa painettu näppäin oikein on. Lukeminen ja kirjoittaminen voidaan tietysti tehdä rivien ja sarakkeiden kannalta toisinkin päin.[6.]

Painetun näppäimen selville saamiseksi kirjoitetaan rivijohtimiin ensin sana 0001, sitten 0010, sitten 0100 ja lopuksi 1000. Jokaisen kirjoituksen jälkeen luetaan sarakejohtimet. Näiden neljän kirjoitus- ja neljän lukujakson jälkeen ohjelmistolla pystytään päättelemään, mikä näppäin on painettuna.[6.]

4.4 Ympäristö

Ohjelma toteutettiin c-kielellä, ja ohjelmointiympäristönä ohjelmoinnissa käytettiin PC-tietokonetta sekä IAR ANSI C -kääntäjää. Ohjelman testaukseen testilaitteessa käytettiin EMUL51-emulaattoria. Ohjelman kehitys- ja testaustyö tapahtui Kajaanin ammattikorkeakoulun tietokonetekniikan laboratoriossa. Laboratoriossa oli kaikki tarvittavat ohjelmistot ja laitteet kehitys- ja testaustyötä varten. Ohjelman koodin ohjelmoiminen tapahtui myös osaksi kotikoneella, jossa ohjelmointiympäristönä käytettiin Microsoft Visual Studio .NET 2003:a.

IAR ANSI C -kääntäjä kääntää korkean tason kielen lähdekoodin mikrokontrollerille ladattavaksi binäärikoodiksi. EMUL51-emulaattorin mahdollistaa täydellisen mikrokontrollerin hallinnan ilman, että ohjaimen resursseja käytettäisiin. Tällöin kaikki keskeytykset, portit ja muisti ovat käytettävissä.

5 JÄTKÄNSHAKKIPELIN TOTEUTUS 8051:LLE

5.1 Ohjelman yleiskuvaus

Ohjelman alkaessa suoritetaan laitteistossa tarvittavat alustukset eli graafisen LCD-näytön alustusrutiinit. Alustusten jälkeen näytölle piirretään 3 x 3 -peliruudukko, jonka jälkeen aloitetaan varsinainen peli. ”Pelaaja 1” aloittaa aina pelin, kun ohjelma käynnistyy. Tietokonepelaajan vaikeustaso on aina pelin alussa 1, eli helpoin mahdollinen.

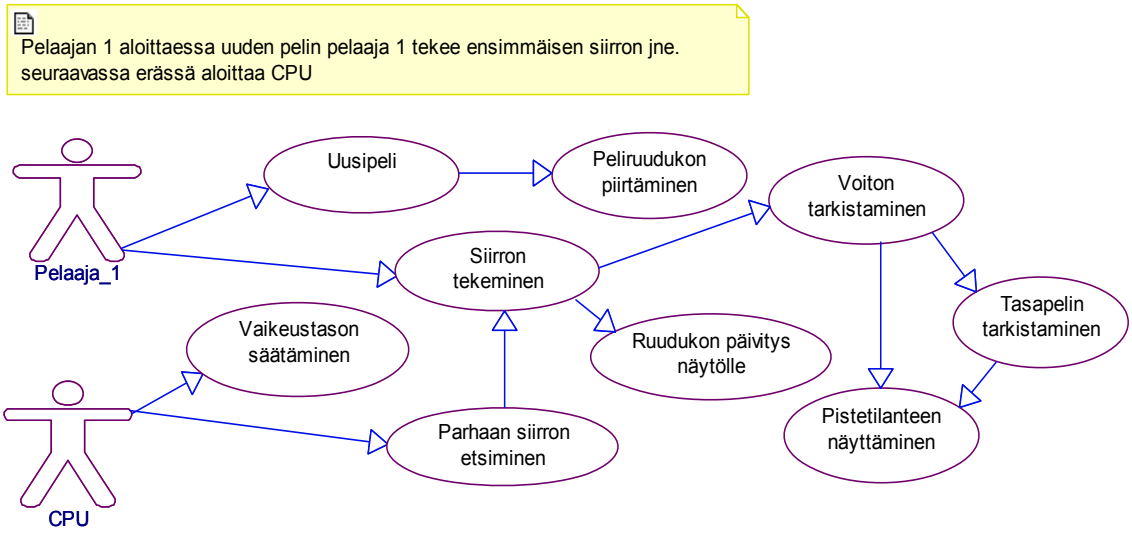
Pelaaja valitsee paikan peliruudukolta matriisinäppäimistön numeronäppäimillä 1-9, johon siirto tehdään. Peliruudukkoa päivitetään näytölle aina, kun jompikumpi on tehnyt siirron. Eli siis piirretään merkki ”X” tai ”O” siirron tekijästä riippuen. Seuraavan siirron tekee luonnollisesti tietokone. Jokaisen siirron jälkeen ”skannataan”, eli käydään läpi koko peliruudukko ja tarkistetaan, onko jompikumpi saanut kolmen suoraa aikaiseksi. Ensimmäinen erä päättyy, kun voittaja on selvillä. Erä päättyy myös silloin, kun pelissä päädytään tasapelitilanteeseen, eli siirtoja on yhteensä tehty 9 ja voittaja ei ole selvillä.

Pelaaja 1 pelaa aina kaksi erää kutakin tietokoneen vaikeustasoa vastaan. Kumpikin siis pääsee aloittamaan kutakin vaikeustasoa kohti. Erän päättyessä tulostetaan näytölle voittaja, jonka jälkeen näytetään kokonaistilanne. Peli päättyy, ja ohjelmassa siirrytään alkuun, kun pelaaja on pelannut kaikki erät eri vaikeustasoilla. Lopuksi ohjelma tulostaa vielä näytölle lopputulokset peleistä. Koko ohjelman periaatteellinen toimintakaavio on esitetty liitteessä 1.

Ohjelmisto on jaettu kahteen ohjelmamoduuliin. Pääohjelma JShakki.c sisältää pelin toiminnallisuuden ja LCD_ohjaus.c LCD-näytön ohjaukseen tarvittavat rutiinit. Lisäksi ohjelmisto sisältää otsikkotiedostot JShakki.h ja LCD_ohjaus.h. Otsikkotiedostot sisältävät aliohjelmien esittelyt sekä joitakin ohjelmistoon liittyviä määrittelyjä.

5.2 Vaatimusten määrittely

Ohjelman suunnittelussa ja määrittelyssä on käytetty osaksi UML-mallinnusta. Kuvassa 6 on esitetty jätkänshakkipelin toiminnallisuus käyttötapauskaavion avulla. Kaaviosta käy ilmi, miten ohjelmassa edetään tiettyjen valintojen ja ohjelmatoimintojen jälkeen.



Kuva 6. Jätkänshakkipelin käyttötapauskaavio

Alla olevissa taulukoissa 2-10 on esitetty ohjelman käyttötapauslomakkeet.

Taulukko 2. Uusi peli

Käyttötapaus:	Uusi peli
Toimijat:	Pelaaja 1
Esiehdot:	LCD-näyttö on alustettu. Ohjelma on käynnistetty, ja se kysyy pelaajalta uuden pelin aloittamista.
Kuvaus:	Ohjelma kysyy käyttäjältä ”Aloita uusi peli painamalla ’F’ ”. Pelaaja aloittaa uuden pelin nappia painamalla.
Poikkeukset:	Ohjelma ei ole käynnissä. Peli on käynnissä ja pelierät ovat kesken. LCD-näytön alustus epäonnistunut.
Lopputulos:	Ohjelma tulostaa näytölle ”aloitetaan uusi peli” ja kysyy siirtoa pelaajalta.

Taulukko 3. Peliruudukon piirtäminen

Käyttötapaus:	Peliruudukon piirtäminen
Toimijat:	CPU
Esiehdot:	Uusi peli on valittu. Tarvittavat laitteistoalustukset on suoritettu.
Kuvaus:	Ohjelma piirtää 3 x 3 -peliruudukon näytölle.
Poikkeukset:	Näyttö ei alustu.
Lopputulos:	Näytöllä on 3 x 3 -peliruudukko näkymä.

Taulukko 4. Siirron tekeminen

Käyttötapaus:	Siirron tekeminen
Toimijat:	CPU tai Pelaaja 1
Esiehdot:	CPU: Tutkittu, monesko erä on menossa ja vaikeustaso säädetty. Paras siirto ”etsitty”. Pelaaja 1: Pelaaja 1:n vuoro tehdä siirto ja laillisia siirtoja on jäljellä.
Kuvaus:	Pelaaja 1 tai CPU tekee siirron, joka päivitetään peliruudukolle.
Poikkeukset:	Pelissä ei ole laillisia siirtoja jäljellä. Erä on loppunut. Kaikki erät on pelattu → peli on loppunut.
Lopputulos:	Pelaaja 1 tai CPU on tehnyt laillisen siirron ja se on päivitetty peliruudukkoon. Vuoro siirtyy toiselle siirrontekijästä riippuen.

Taulukko 5. Voiton tarkistaminen

Käyttötapaus:	Voiton tarkistaminen
Toimijat:	CPU
Esiehdot:	CPU tai Pelaaja 1 on tehnyt laillisen siirron peliruudukkoon.
Kuvaus:	Käydään läpi peliruudukko ja tutkitaan, onko ruudukolla kolmen merkin suoria.
Poikkeukset:	Siirtoa ei ole tehty. Peli on loppunut.
Lopputulos:	Ruudukko on käyty läpi ja ilmoitettu voitosta, mikäli kolmen suoria on löytynyt, muutoin jatketaan peliä eteenpäin.

Taulukko 6. Ruudukon päivitys näytölle

Käyttötapaus:	Ruudukon päivitys näytölle
Toimijat:	CPU
Esiehdot:	Peli aloitettu ja siirto tehty.
Kuvaus:	Piirretään merkki ”X” tai ”O” peliruudukkoon siirron tekijästä riippuen.
Poikkeukset:	LCD-näyttö ei ole alustunut. Siirtoa ei ole tehty. Laillisia siirtoja ei ole jäljellä.
Lopputulos:	Merkki ”X” tai ”O” on päivittynyt näytölle valittuun kohtaan.

Taulukko 7. Vaikeustason säätäminen

Käyttötapaus:	Vaikeustason säätäminen
Toimijat:	CPU
Esiehdot:	Peli on aloitettu. Kaksi erää pelattu edellistä vaikeustasoa kohti.
Kuvaus:	Pelin alkaessa vaikeustaso on helpoin. Vaikeustasoa säädetään vaikeammaksi, kun kaksi erää pelattu kutakin vaikeustasoa kohden.
Poikkeukset:	Eriä vielä pelaamatta vaikeustasoa kohden. Peli on päättynyt, eli vaikein taso pelattu loppuun.
Lopputulos:	CPU on säätänyt vaikeustasoa vaikeammaksi ja kysyy pelaajalta siirtoa.

Taulukko 8. Parhaan siirron etsiminen

Käyttötapaus:	Parhaan siirron etsiminen
Toimijat:	CPU
Esiehdot:	Laillisia siirtoja jäljellä. CPU:n vuoro siirtää.
Kuvaus:	CPU tekee siirron pelitilanteen ja vaikeustason mukaan. Eri vaikeustasoilla on erilaisia käytäntöjä siirron tekemiseksi. Esim. siirto voi olla puolustus-, hyökkäys- tai aloitussiirto.
Poikkeukset:	Laillisia siirtoja ei ole jäljellä. Peli on pelattu loppuun.
Lopputulos:	CPU on tehnyt parhaan siirron omalta kannalta vaikeustasosta riippuen tai pelitilanteesta riippuen. Kysytään pelaajalta seuraavaa siirtoa.

Taulukko 9. Pistetilanteen näyttäminen

Käyttötapaus:	Pistetilanteen näyttäminen
Toimijat:	CPU
Esiehdot:	Pelattavia eriä on jäljellä. Erä on saatu päätökseen.
Kuvaus:	Näytetään erät, voitot, häviöt ja tasapelit käynnissä olevasta pelistä.
Poikkeukset:	Pelattavia eriä ei ole jäljellä → peli päättynyt. Pelattava erä on kesken.
Lopputulos:	Näytetään näytöllä pistetilanne tällä hetkellä, eli pelattujen erien määrä, voitot, häviöt ja tasapelit.

Taulukko 10. Tasapelin tarkistaminen

Käyttötapaus:	Tasapelin tarkistaminen
Toimijat:	CPU
Esiehdot:	CPU tai Pelaaja 1 on tehnyt laillisen siirron peliruudukkoon.
Kuvaus:	Voittaja ei ole selvillä ja siirtoja on tehty yhteensä 9.
Poikkeukset:	Siirtoa ei ole tehty. Peli on loppunut. Toinen pelaajista on voittanut pelin.
Lopputulos:	Voittaja ei ole selvillä ja siirtoja tehty 9 ilmoitetaan tasapelistä. Muussa tapauksessa jatketaan peliä.

5.3 Laitteistossa tarvittavat ohjelmat

5.3.1 LCD-näyttö

Ennen kuin LCD-näyttöä voidaan käyttää ohjelmistossa, joudutaan se alustamaan. Näytön alustus tapahtuu kirjoittamalla alustustavut näytön kontrolli- ja datarekisterin osoitteisiin. Taulukossa 10 on esitetty LCD-näytön rekisterit. Rekistereihin kirjoitettavat alustustavut on esitetty taulukossa 11.

Taulukko 11. LCD-näytön rekisterit.

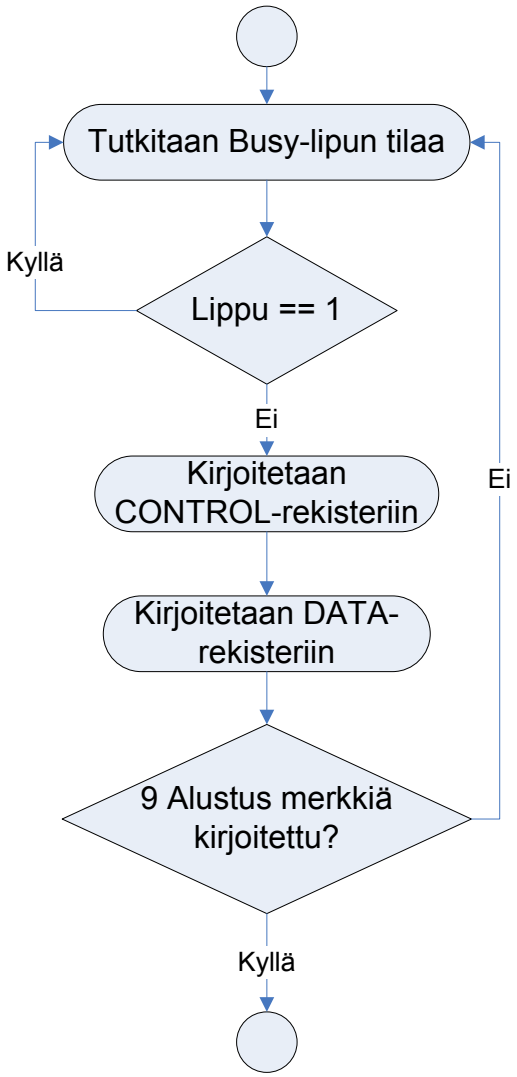
Rekisteri	Osoite
Control	0xC002
Data	0xC000

Taulukko 12. LCD-näytön rekistereihin kirjoitettavat alustustavut.

Alustus	CONTROL	DATA
Mode Control	0x00	0x32
Character Pitch	0x01	0x77
Horizontal Count	0x02	0x1F
Multiplex Ration	0x03	0x3F
Cursor Position	0x04	0x07
Display Start Address (Low)	0x08	0x00
Display Start Address (High)	0x09	0x00
Cursor Address (Low)	0x0A	0x00
Cursor Address (High)	0x0B	0x00

Taulukossa 11 esitetyillä alustustavuilla alustetaan LCD-näyttö grafiikkatilaan: 256 x 128. 1-bittisessä grafiikkatilassa LCD-näytön ruudun leveys on 256 pikseliä ja korkeus 128. Ohjelmistossa käytetään myös näytön tekstitilaa, jonka vuoksi näyttö alustetaan välillä tekstitilaan. Tällöin datarekisteriin kirjoitettavat tavut ovat hieman poikkeavat grafiikkatilan alustuksista.

Alustus tapahtuu kuvassa 7 esitetyn vuokaavion mukaisesti. Alustusrutiinin alussa tarkastellaan BUSY-lipun tilaa. Tarkastelun perusteella voidaan todeta, onko näyttö varattu vai voidaanko rekistereihin kirjoittaa. BUSY-lippu on kahdeksas bitti osoitteessa 0xC003. Mikäli bitti on tilassa 1, on näyttö varattu, muussa tapauksessa näytölle voidaan kirjoittaa.



Kuva 7. LCD-näytön alustusrutiini.

Kun alustusrutiini on suoritettu ja kaikki 9 alustusmerkkiä on kirjoitettu rekistereihin onnistuneesti, LCD-näyttö on toimintavalmis.

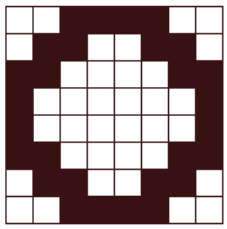
5.3.2 Näytölle piirtäminen

Näytön alustusten jälkeen näytölle voidaan piirtää tai kirjoittaa tekstiä, riippuen siitä, mihin tilaan näyttö on alustettu. Näytölle piirtäminen aloitetaan asettamalla CONTROL-rekisteriin arvo 0x0C. Tällöin näyttö tietää, että DATA-linjoilta tuleva data ohjataan suoraan näytölle. DATA-rekisteriin kirjoitettava data menee aina siihen kohtaan, mihin kursori on kulloinkin asetettu.

Ohjelmassa piirretään tarvittavat kuvat näytölle tietyillä rutiineilla, jotka on toteutettu omiksi aliohjelmiksi. Aliohjelmat sijaitsevat LCD-näytön ohjausohjelmamoduulissa. Kuva muodostuu 8 bitin jaksoista eli tavuista. Tavut voidaan kirjoittaa heksalukuina näytölle tavu kerrallaan DATA-rekisteriin. Esimerkiksi heksaluku 0xAA sytyttää näytölle kuvapisteitä siten, että joka toinen pikseli on päällä (0xAA = 1010 1010B).

Ohjelman kuvat on toteutettu const char -taulukoiksi, jotka sisältävät tietyt tavut, joilla kuvat saadaan aikaiseksi. Piirtäminen tapahtuu ohjelmistoon toteutettujen piirtämisrutiineiden avulla. Alla on esitetty esimerkki merkin ”O” piirtämisestä.

```
const char circle[]={0x3C, 0x66, 0xC3, 0x81, 0x81, 0xC3, 0x66, 0x3C};
```



Kuva 8. Merkin O muodostuminen const char -taulukosta.

LCD-näytön ohjelmamoduuli sisältää myös seuraavat ohjelmatoiminnot: LCD-näytön alustus, piirtokursorin paikan asettaminen, näytön tyhjennys- ja viivealiohjelmat.

5.3.3 Matriisinäppäimistö

Näppäimen painaminen näppäimistöltä aiheuttaa keskeytyksen 8051:lle. Keskeytys täytyy sallia kirjoittamalla erillinen käsky IE-rekisteriin. Keskeytykseen tulee myös määrittää, reagoiko keskeytys nousevaan vai laskevaan reunaan. Alla on esitetty ohjelmassa tarvittavat ohjelmarivit.

```
TCON=0x01;  /* INT0 reagoi laskevaan reunaan */
IE=0x81;    /* keskeytys INT0 sallittu */
```

Kun keskeytys tapahtuu, tarkoittaa se ohjelmassa siirtymistä keskeytysaliohjelmaan. Keskeytysaliohjelma on valmis funktio, joka sijaitsee io51.h kirjastossa.

```
interrupt void EX0_int()
{
valinta = nappis & 0x0F;  /* maskataan 4 ylintä bittiä */}
```

Valinta-muuttuja on globaalimuuttuja, johon tietyn näppäimen painamisesta tuleva arvo sijoitetaan. Ohjelma on suunniteltu niin, että kun on pelaaja 1:n vuoro tehdä siirto, ohjelma jää odottamaan valinta arvon muuttumista. Tämän jälkeen ohjelma siirtyy KEYB()-aliohjelmaan, jossa käyttäjän valinta tulkitaan switch-case-rakenteella ja toimitaan sen mukaan. Peliruudukossa on 1…9 paikkaa, johon siirto voidaan tehdä. Näppäimistöllä näppäimien numerot vastaavat peliruudukon paikkoja.

5.4 Jätkänshakkipelin pääohjelma

Pääohjelma on suunniteltu niin, että se pyörii ikuisessa silmukassa. Ohjelman alkaessa se antaa ilmoituksen ”Aloita peli painamalla näppäintä ’F’ ”. Pelin päättyessä, eli kun kaikki erät on pelattu, ohjelma siirtyy alkutilaan. Ohjelma on siis ns. automaattinen. Peli on jaettu eri vaikeustasoihin ja eriin. Pääohjelma sisältää kaikki tarvittavat peliin liittyvät toiminnat. Ohjelmistossa on pyritty toteuttamaan kaikki tärkeimmät toimintarutiinit omiksi aliohjelmiksi.

Varsinaisen pelin alkaessa näytölle piirretään peliruudukkonäkymä. Jokaisen uuden pelin aloittaja on aina pelaaja 1. Tämän jälkeen siirtovuoro vaihtuu luonnollisesti tietokonepelaajalle. X-merkki on määritelty pelaaja 1:n merkiksi ja O-merkki tietokonepelaajan merkiksi.

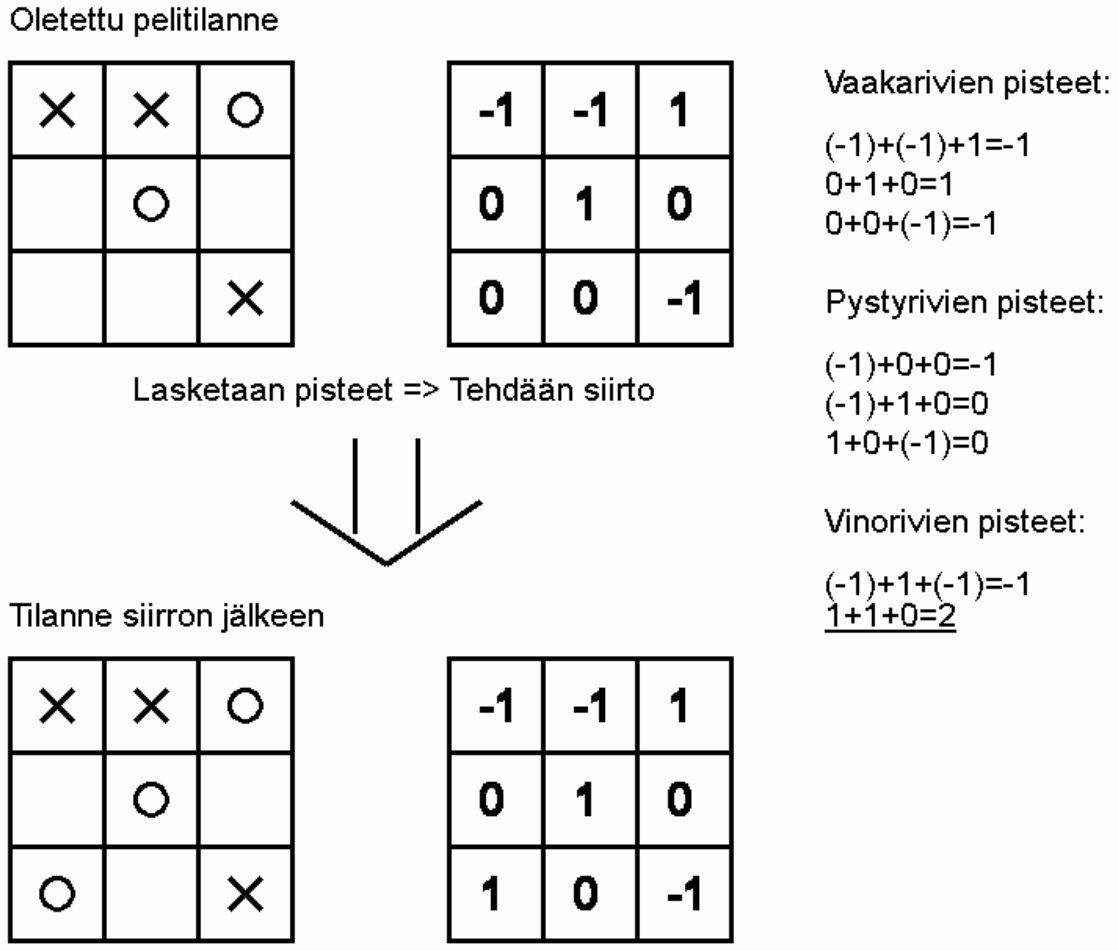
Ohjelmassa on kaksiulotteinen kokonaislukutaulukko, johon pelin siirrot tallentuvat. Ohjelmassa on määritelty pelaaja 1:n merkiksi arvo -1 ja tietokonepelaajan arvoksi +1. Näiden arvojen perusteella voidaan tutkia, kumpi pelaajista on voittaja. Ohjelma tarkistaa jokaisen siirron jälkeen, onko jompikumpi pelin voittaja. Mikäli voittaja ei ole selvillä, tarkastellaan, onko tilanne tasapeli, eli onko siirtoja tehty 9. Pelierää jatketaan niin pitkään, kunnes voittaja tai tasapelitilanne on saavutettu.

Pelierän päättyessä näytölle tulostetaan pistetilanne voitoista ja tasapeleistä. Pelieriä pelataan kutakin vaikeustasoa kohti kaksi, joten kumpikin pelaaja pääsee aloittamaan pelin kullakin vaikeustasolla. Peli päättyy, jos kaikki vaikeustasot on pelattu läpi. Ohjelmassa tutkitaan jokaisen erän alussa pelattujen erien määrä, jonka perusteella vaikeustasoa säädetään automaattisesti. Koko pelin päättyessä ohjelma tulostaa näytölle lopputulokset sekä nollaa kaikki tarvittavat muuttujat ja siirtyy ohjelman alkuun.

Ohjelmassa on omat aliohjelmat, joilla hoidetaan siirtotapahtumaa, estetään samaan paikkaan siirtämistä, näytetään pistetilanne, ja tekoäly, joka on jaettu useampaan osaan. Liitteessä 1 on esitetty periaatteellinen toimintakaavio ohjelmatoiminnasta.

5.5 Jätkänshakkipeliohjelman tekoälyosio

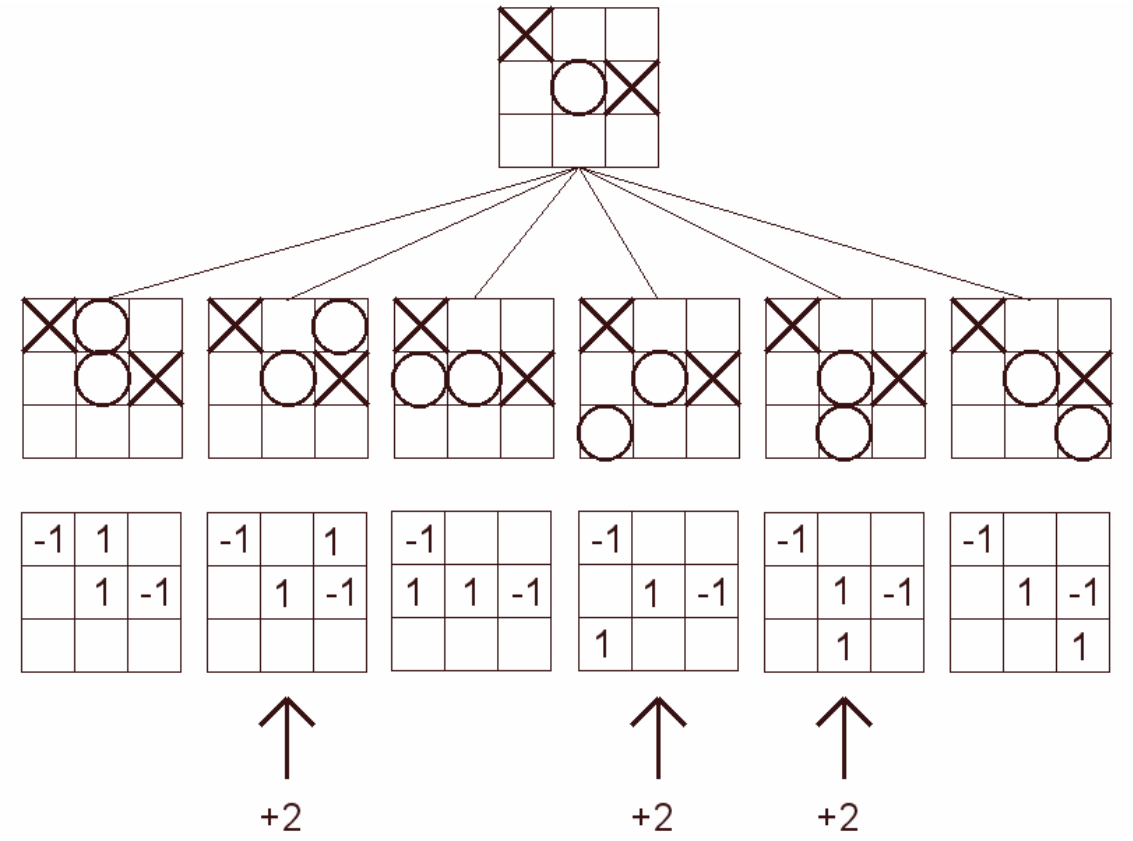
Ohjelman tekoäly perustuu pisteytykseen. Pelaaja 1:n siirroille annetaan siirtoa kohti arvo -1 ja tietokoneelle arvo +1. Näiden arvojen perusteella voidaan tutkia, millainen pelitilanne kulloinkin on ja tehdä sopiva siirto vaikeustasosta riippuen. Yhteen laskemalla pisteitä peliruudukolla voidaan selvittää onko mahdollista rakentaa esim. kolmen suoria. Samalla periaatteella voidaan torjua pelaaja 1:n kahden suoria. Kuvassa 9 on esitetty esimerkki ns. hyökkäyssiirrosta.



Kuva 9. Pisteytyksen periaate.

Peliruudukko käydään läpi, eli jokainen rivi tutkitaan pysty- ja vaakasuunnassa. Merkkien antamat pisteet lasketaan yhteen jokaiselta riviltä. Esimerkissä etsitään omia kahden suoria, eli pisteitä +2. Tässä pelitilanteessa kyseinen pistemäärä löytyy, joten siirto tehdään luonnollisesti kuvan mukaisesti. Kyseisessä tilanteessa tietokone voittaa.

Pisteytysideaa käytetään myös tilanteessa, jossa tietokonepelaajalla tai vastustajalla ei ole kahden suoria, jolloin tällainen joudutaan rakentamaan. Oletetaan, että ”pelaaja 1” on aloittanut pelin ja hänen merkkinsä on ”X” ja tietokonepelaajan ”O”. Peliä on pelattu kolmen siirron verran eteenpäin ja päädytty kuvan 10 mukaiseen pelitilanteeseen.



Kuva 10. Kahden suoran rakentaminen.

Tietokonepelaajalla on tässä tapauksessa 6 kappaletta eri siirtovaihtoehtoja, jotka on esitetty yllä olevassa kuvassa. Tässä kohtaa käytetään pelipuuideaa, eli käydään kaikki mahdolliset siirrot läpi hakusyvyydellä 1. Pelitilanteista yritetään löytää +2:n suoria.

Laskenta tapahtuu samalla tavalla, kuten kuvassa 9 on esitetty. Ohjelmassa tämä idea toimii siten, että peliruudukolle tehdään siirto, lasketaan pisteet, merkitään siirtopaikka ylös ja perutaan siirto. Tämän jälkeen tehdään seuraava siirto jne. Kuvan 10 pelitilanteessa löydetään kolme tilannetta, joissa päädytään pisteisiin +2. Siirto siis valitaan näitten tilanteiden joukosta.

Ohjelma sisältää joukon tekoälyyn liittyviä aliohjelmia, joita käytetään siirron tekemiseen tietyissä pelitilanteissa. Tekoäly on jaettu eri vaikeustasoihin ja tekoälyn aliohjelmia käytetään vaikeustason mukaan. Vaikeustasoa kasvatetaan, kun pelaaja on pelannut kaksi erää yhtä tasoa kohti. Seuraavissa luvuissa on selitetty tarkemmin eri vaikeustasojen toimintaperiaatteet.

5.5.1 Vaikeustaso 1

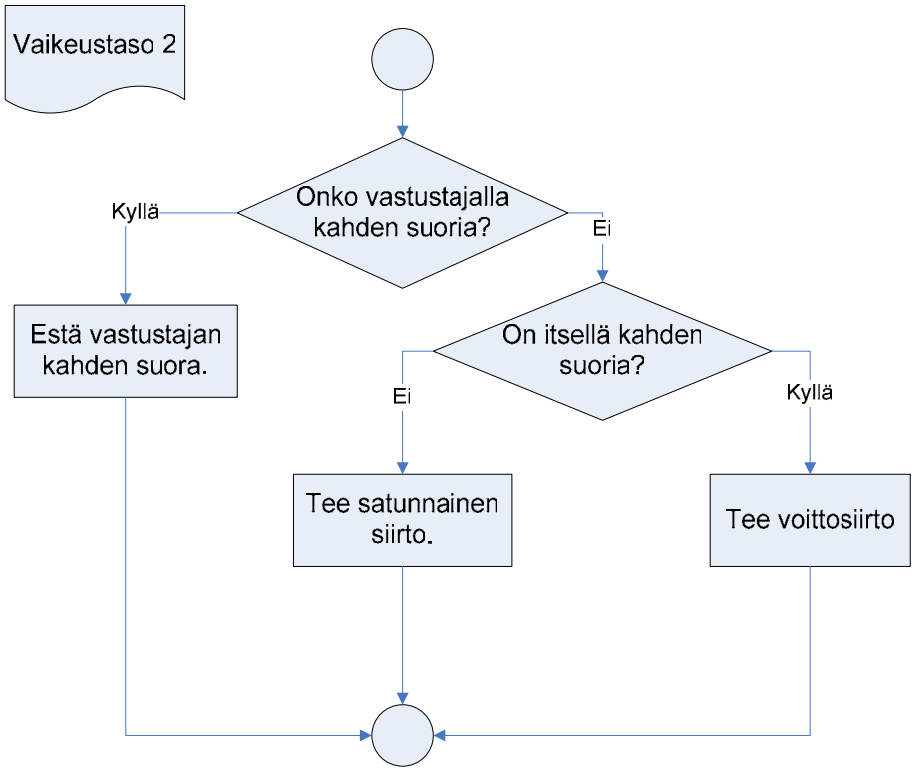
Vaikeustasossa 1 oletetaan, että tietokone osaa vain pelin säännöt, mutta ei osaa pelata kovinkaan hyvin. Tietokone on siis tällä tasolla aloittelija, ja se pelaa peliä ”ensimmäistä kertaa”. Pelin alkaessa vaikeustaso on säädetty tälle tasolle.

Tietokone ei osaa analysoida pelaajan tekemiä siirtoja mitenkään, eikä se osaa myöskään reagoida niihin. Tietokoneen siirto on siis satunnainen valinta peliruudukolta. Tällä tasolla tietokonepelaaja voi voittaa ainoastaan pelkällä tuurilla, jos oletetaan, että vastustajakin on aloittelija.

5.5.2 Vaikeustaso 2

Vaikeustasolla 2 tietokonepelaaja on hieman aloittelijaa kokeneempi, mutta ei kuitenkaan vielä kokenut pelaaja. Tällä tasolla tietokonepelaajan ensisijainen pyrkimys on estää vastustajan tekemät kahden suorat ja voittomahdollisuudet. Tietokone ei osaa tehdä itselleen tässä vaiheessa vielä kahden, suoria vaan se voi saada sellaisen sattumalta. Mikäli pelissä

päädytään tilanteeseen, jossa tietokoneella on kahden suora ja vuoro on tietokoneella, ”näkee” se tämän suoran ja tekee voittosiirron. Tietokonepelaajan aloitussiirrot perustuvat satunnaisiin siirtoihin peliruudukolta. Siirto on satunnainen myös silloin, kun vastustajalla tai itsellä ei ole kahden suoria. Kuvassa 11 on esitetty tekoälyn toimintalogiikka vuokaavion avulla.



Kuva 11. Vaikeustason 2 toimintalogiikka.

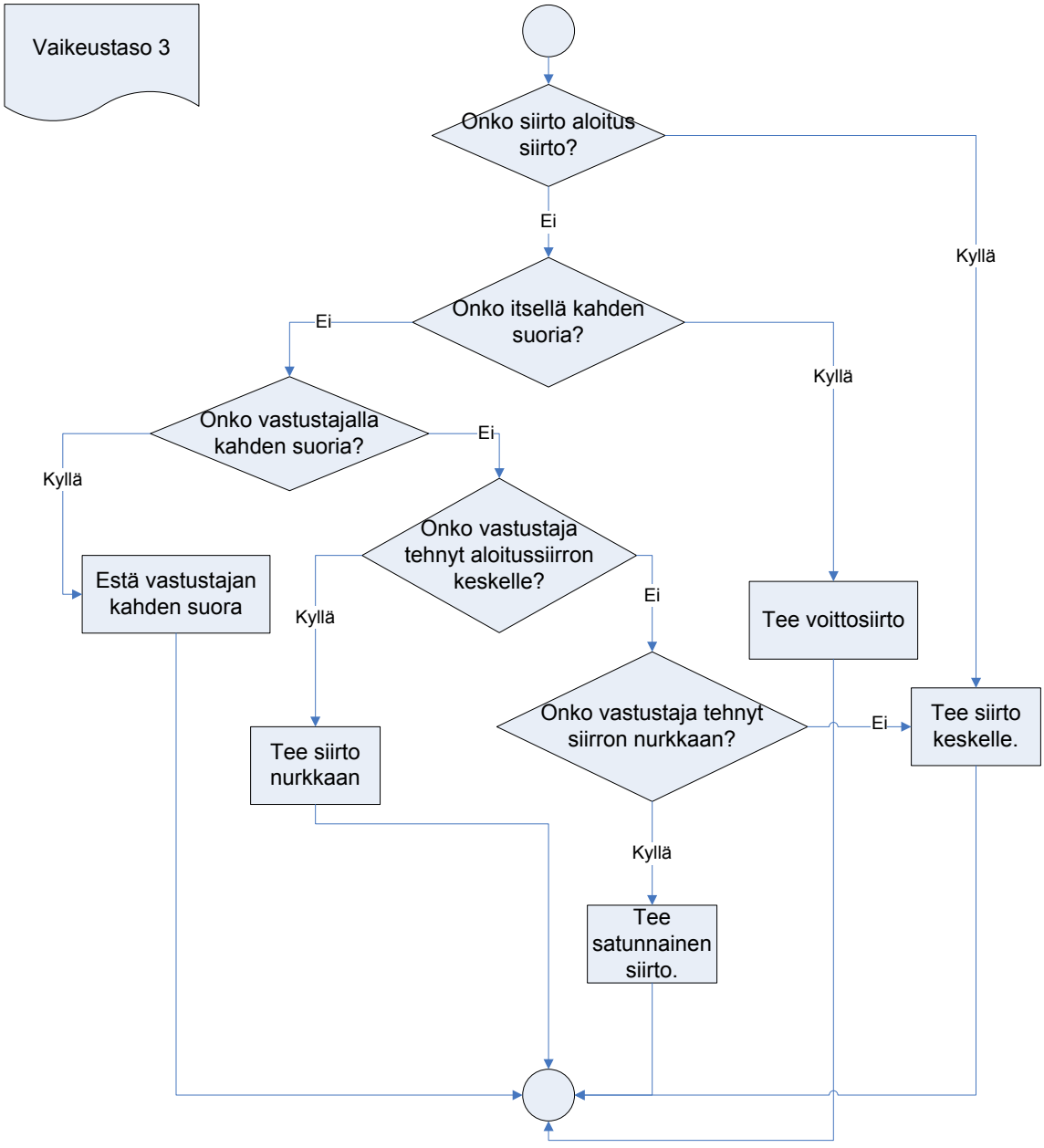
5.5.3 Vaikeustaso 3

Vaikeustasolla 3 tietokonepelaajan oletetaan olevan jo kokenut pelaaja. Tällä tasolla tietokonepelaaja tietää aloitussiirron merkityksen ja pyrkii toimimaan sen mukaan seuraavassa siirrossaan. Mikäli vastustaja tekee siirron keskimmäiseen peliruutuun, tekee tietokonepelaaja nurkkasiirron. Vastustajan tehdessä nurkkasiirron tekee tietokonepelaaja siirron keskelle. Myös tietokonepelaajan tekemä aloitussiirto on aina keskellä.

Tälläkään tasolla tietokonepelaaja ei osaa vielä rakentaa kahden suoria, vaan mikäli itsellä tai vastustajalla ei ole kahden suoria tehdään satunnainen siirto. Kuitenkin kahden suora

muodostuu tällä tasolla aina, kun tietokonepelaaja aloittaa pelin, koska aloitussiirto tehdään keskelle.

Pelin edetessä eri pelitilanteisiin pyrkii tietokone esisijaisesti voittamaan vastustajan, eli rakentamaan kahden suorasta kolmen suoran. Toissijaisena tavoitteena on vastustajan kahden suorien estäminen eli voittosiirron tekeminen. Kuvassa 12 on esitetty tekoälyn toimintalogiikka vuokaavion avulla.



Kuva 12. Vaikeustason 3 toimintalogiikka.

5.5.4 Vaikeustaso 4

Tällä tasolla tietokone on oppinut pelin eri strategiat ja jipot. Tietokonetta ei ole mahdollista voittaa tällä tasolla, vaan peli päättyy aina tasan tai tietokone voittaa.

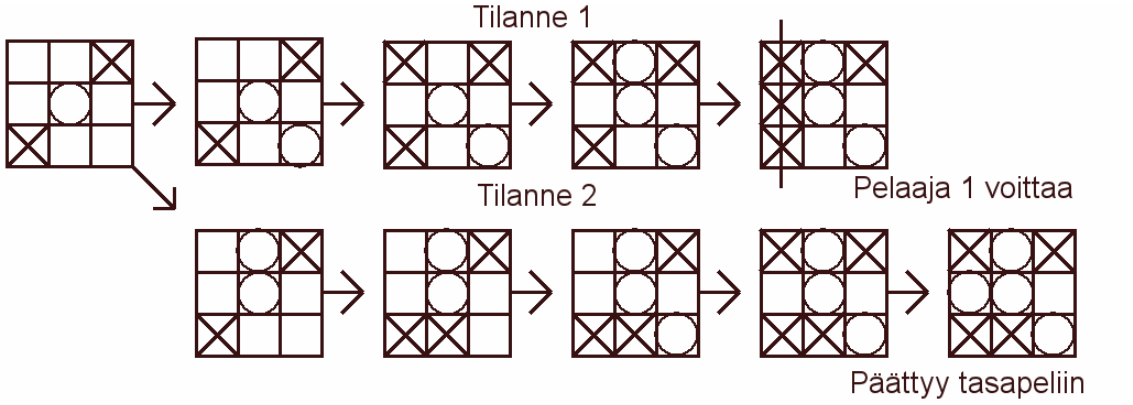
Peliruudukon paikoilla on tietty siirtojärjestys, jota käydään läpi silloin, kun kyseessä on tietokonepelaajan ensimmäinen tai toinen siirto. Tällöin huomioidaan aloitussiirron merkitys. Mikäli tietokonepelaaja aloittaa pelin, siirtää se aina keskelle ruudukkoa. Vastustajan aloittaessa toimitaan kuten vaikeustasolla 3. Eli jos vastustaja tekee siirron keskimmäiseen peliruutuun, tekee tietokonepelaaja nurkkasiirron. Vastustajan tehdessä nurkkasiirron tekee tietokonepelaaja siirron keskelle. Tämä on kuitenkin toteutettu hieman eri tavalla ohjelmassa tällä vaikeustasolla. Kuvassa 13 on esitetty peliruudukon siirtojärjestys.

2	6	4
7	1	8
5	9	3

Kuva 13. Peliruudukon paikkojen siirtojärjestys.

Ensisijainen tarkoitus tietokonepelaajalla on voittaa vastustaja, eli se etsii mahdollisia kahden suoria, joista voisi rakentaa kolmen suoria. Seuraavana tavoitteena on vastustajan kahden suorien estäminen eli voittosiirron tekeminen.

Mikäli edellä mainittuja tilanteita ei havaita, katsotaan, yrittääkö vastustaja luoda tiettyä strategiaa, jolla se voisi voittaa pelin. Tällä vaikeustasolla on yksi strategia, millä pelin voisi voittaa, jos tätä ei otettaisi huomioon ajoissa. Tämä strategia on tietysti otettu huomioon tällä vaikeustasolla, joten tämäkään tilanne ei johda vastustajan voittoon. Kuvassa 14 on esitetty tilanne, joka johtaisi vastustajan voittoon. Alla olevassa kuvassa näytetään myös, kuinka tilanne estetään.

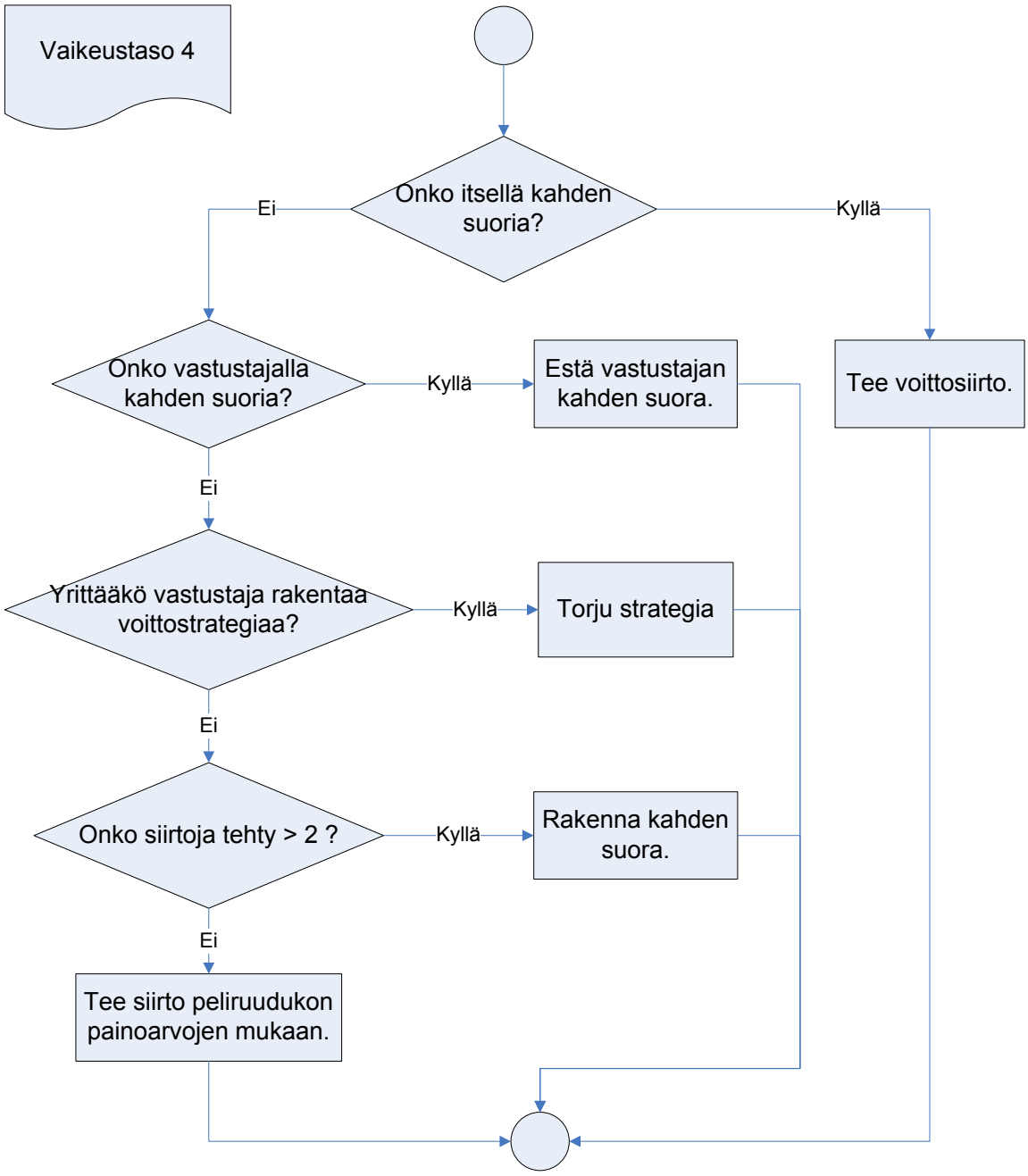


Kuva 14. Vastustajan luoma voittoon johtava strategia.

Oletetaan, että pelaaja 1 on tehnyt ensimmäisen siirron nurkkaan ja pelaajan merkki on ”X”. Tietokonepelaaja tekee siirron keskelle. Seuraavaksi pelaaja 1 tekee siirron vastakkaiseen nurkkaan. Mikäli tässä vaiheessa ei otettaisi huomioon kyseistä strategiaa, saattaisi pelitilanne johtaa kuvassa olevaan tilanteeseen 1. Tällöin siis pelaaja 1 voittaa pelin.

Tilanteessa 2 tietokonepelaaja on huomioinut tilanteen ja tekee siirron kuvan mukaisesti. Tässä tapauksessa vastustajan on estettävä tietokonepelaajalle muodostunut kahden suora tai muutoin tietokone voittaa. Tässä tilanteessa peli tulee todennäköisesti päättymään tasapeliin.

Kuvassa 15 esitettyä vuokaaviota edetessä voidaan päätyä myös sellaiseen pelitilanteeseen, että mikään edellä mainituista pelitilanteista ei toteudu. Tällöin yritetään rakentaa kahden suora. Mikäli tämäkään ei ole mahdollista, tehdään siirto peliruudukon paikkojen painoarvojen mukaan vapaaseen paikkaan. Luultavimmin tässä tilanteessa kummallakaan ei ole mahdollisuutta voittoon, vaan peli päättyy tasan. Kuvassa 15 on esitetty tekoälyn toiminta vuokaavion avulla.



Kuva 15. Vaikeustason 4 toimintalogiikka.

5.6 Pelipuut ja MinMax-algoritmi

Klassisten strategiapelien de facto -menetelmä on haku MinMax-puussa. Ne kuvaavat pelin eri tiloja, joiden solmut vastaavat pelaajien mahdollisia siirtoja. Jokaisella puun tasolla on yhden pelaajan yhden vuoron siirtomahdollisuudet. Puussa olevilla solmuilla on arvot, ja ne vastaavat pelin ”läheisyyttä” voitosta.[4.]

MinMax-puussa eteneminen tapahtuu seuraavasti: ensimmäinen pelaaja valitsee solmun, joka maksimoi pisteet, tämän jälkeen toinen pelaaja valitsee valitun solmun lapsen, joka minimoi pisteet. Algoritmissa siis oletetaan, että vastustaja pyrkii vain minimoimaan pisteet jatkuvasti. MinMax-puut voidaan laittaa toimimaan peleihin, joissa sattumalla on vaikutusta, esimerkiksi nopan heitto voisi olla tällainen tapaus. Silloin se eroaisi MinMax-puusta siten, että pelaajien siirtoja vastaavien solmujen väliin lisättäisiin ns. sattumasolmuja, joiden avulla satunnaisuus voitaisiin ottaa huomioon.[4.]

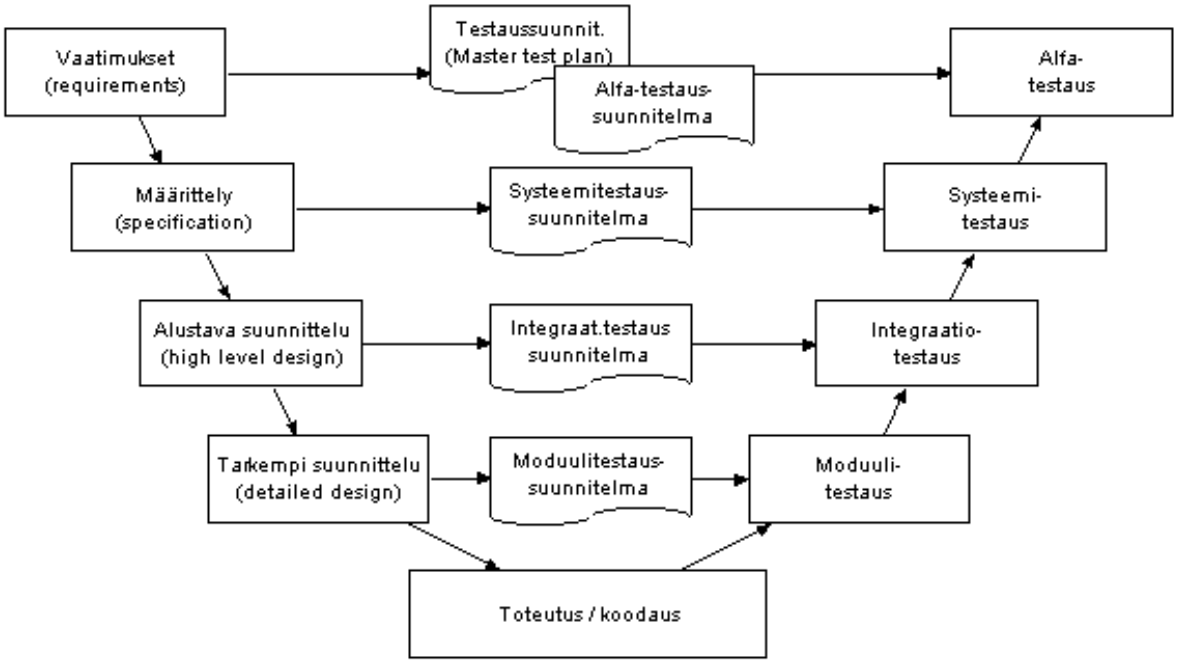
Jo von Neumann todisti, että käymällä koko MinMax-puun läpi pelaaja löytää parhaimman mahdollisen siirtoyhdistelmän itselleen. Huonona puolena minmax-puissa on se ominaisuus, että ne voivat olla valtavan isoja, tietenkin riippuen siitä, kuinka paljon mahdollisia siirtoja aina missäkin tapauksessa on. Esimerkiksi shakissa on vuorollaan yleensä noin 35 siirtoa. Silloin jo kolmen siirtoparin syvyisessä minmax-puussa on melkein kaksi miljardia (356) solmua.[4.] Tällöin aikaa kuluisi näin yksinkertaisella menetelmällä aivan liian paljon siirron tekoon.

Minmax-puiden hakuavaruutta on kuitenkin mahdollista yrittää rajoittaa ns. alfa-beta-karsinnalla. Alfa-beta-karsinta pystyy parhaimmillaan vähentämään min-max-puun haarojen määrän neliöjuureen alkuperäisestä. Esimerkiksi shakin tapauksessa se tarkoittaisi solmujen lasten lukumäärän vähenemistä 35:stä kuuteen. Alfa-beta-karsinta ei kuitenkaan pysty takaamaan pahimman tapauksen aikavaativuutta, eli joillain minmax-puilla se ei nopeuta puun läpikäymistä lainkaan.[4.]

5.7 Testaus

5.7.1 Testausmenetelmät

Testaus on suoritettu V-mallin mukaisesti, joka on esitetty kuvassa 16. Eri testitapaukset on suunniteltu vaatimusten määrittelyn pohjalta. Ohjelmiston moduulit (aliohjelmat) testataan ohjelmointi vaiheessa. Tämän jälkeen suoritetaan integrointitestaus, jossa testataan moduulien ja osajärjestelmien väliset liittymät. Lopuksi suoritetaan järjestelmätestaus, jossa testauksen kohteena on koko järjestelmä. Tässä vaiheessa katsotaan, vastaako ohjelmiston toiminta määrittelyä.



Kuva 16. Testauksen V-malli. [7.]

Moduulitestaus (Module testing)

Moduulitestaus (yksikkötestaus, unit testing) käsittää pienimpien ohjelmayksiköiden, esimerkiksi funktioiden, ja niistä muodostuvien pienten kokoelmien eli moduulien testauksen. Moduulitestauksen tavoitteena on löytää selvien virheiden lisäksi ristiriitoja yksiköiden ja moduulien määrittelyjen ja toiminnan välillä sekä korjata ne. [7.]

Integraatiotestaus (Integration testing)

Integraatiotestauksella tarkoitetaan vaihetta, jossa pienemmät osat kootaan yhteen ja testataan, kunnes koko järjestelmä on saatu kokoon. Tarkoituksena on siis yhdistellä moduuleita tai moduuliryhmiä (osajärjestelmiä). Integraatiotestauksen tarkoituksena on varmistaa, että yhdistetyt osat toimivat määritellysti keskenään. Erityisesti tarkkailun kohteena ovat moduulien väliset rajapinnat. Testauksessa on tarkoitus löytää ja korjata mahdollisimman paljon yhteensoveltumattomia moduuleita.[7.]

Järjestelmätestaus (Systeemitestaus (System testing))

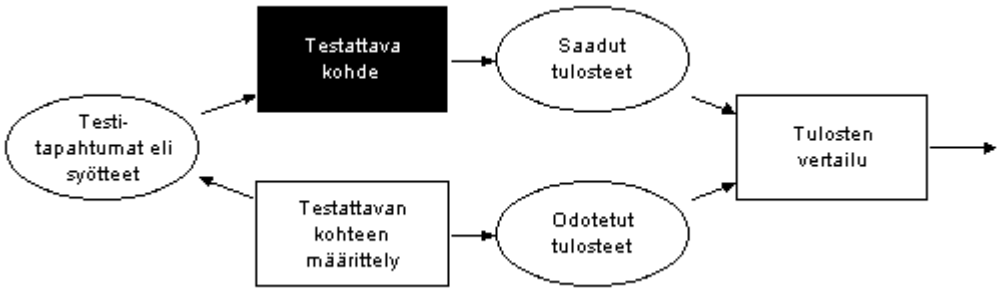
Järjestelmätestauksessa tarkastelun kohteena on koko järjestelmä. Testauksen päätavoite on tutkia, täyttääkö valmis systeemi (esim. ohjelmisto) sen määrittelyn asettamat vaatimukset. Toisaalta tavoitteena voidaan taas pitää sitä, että löydetään ja korjataan mahdollisimman paljon ristiriitoja systeemin ja sen määrittelyn väliltä. Joskus voi olla lisäksi tarpeellista testata esimerkiksi valmiin ohjelman suorituskykyä, rasituskestävyyttä, turvallisuutta ja toimivuutta verkossa.[7.]

Dynaaminen analyysi

Testauksessa on käytetty dynaamista analyysiä, jossa keskitytään ohjelman ja koodin käyttäytymiseen suorituksen aikana. Testaus edellyttää toimivaa ohjelmaa tai ympäristön, missä esimerkiksi yksittäisiä funktioita voidaan suorittaa. Dynaamisella analyysillä saadaan testattua ominaisuuksia, jotka staattisessa analyysissä olisivat mahdottomia tai ainakin vaikeita testata. [7.] Testauksessa on käytetty yleisimpiä dynaamisen analyysin testaus-strategioita, joita ovat Black Box -testaus ja White Box -testaus.

Black box -testaus

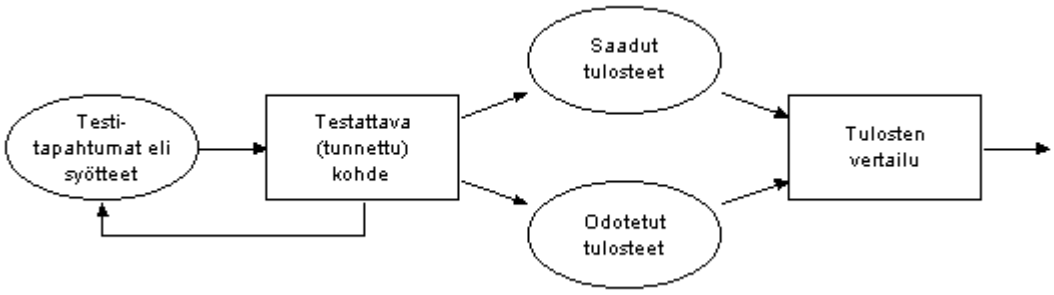
Black box -testaus perustuu testattavan systeemin (esim. ohjelma) tai komponentin (esim. funktio) input-output-käyttäytymiseen. Black box -testauksessa ei välitetä testattavan kohteen rakenteesta tai sisällöstä, vaan tutkittavana ovat kohteen tulosteet (output) erilaisilla syötearvoilla (input). Testaajalle kohde on siis jokin tuntematon "musta laatikko", black box. Testattavan kohteen oikeellisuutta tarkastellaan vertaamalla saatuja tulosteita haluttuihin tai odotettuihin tulosteisiin. Testitapaukset johdetaan aina kohteen määrittelyn perusteella.[7.]



Kuva 17. Black Box -testaus.[7.]

White box-testaus

Päinvastoin kuin black box -testauksessa, jossa testitapaukset johdetaan kohteen määrittelystä, white box -testauksessa testitapaukset johdetaan kohteen, esimerkiksi koko ohjelman, sisäisestä rakenteesta ja logiikasta. Tavoitteena on valita testitapaukset siten, että kaikki kohteen (esimerkiksi ohjelman tai funktion) haarat ja ohjelmapolut tulisi käytyä läpi.[7.]



Kuva 18. White box -testaus.[7.]

5.7.2 Testauksen toteutus

Ohjelma on toteutettu kahdessa osassa. Ensiksi ohjelmasta on toteutettu PC-versio, joka toimi ainoastaan PC-tietokoneella. Toiseksi osaksi ohjelmaa on toteutettu sulautetun järjestelmän tarvitsemat ohjelmat, kuten näytön sekä näppäimistön toiminnallisuus. Tämän jälkeen PC-versio on tehty yhteensopivaksi käytetyn laitteiston kanssa ja ohjelmamoduulit on yhdistetty yhdeksi ohjelmistoksi. Ohjelmasta on tehty PC-versio siitä syystä, että itse pelin toiminnallisuutta olisi helpompi testata käytännössä. Tästä syystä testauksessa on käytetty kahta erilaista testausympäristöä.

PC-version testauksessa käytettiin testausympäristönä Microsoft Visual Studio .NET 2003:a. Testilaitteen kanssa testausympäristönä käytettiin PC-tietokonetta, IAR ANSI C -kääntäjää, EMUL51-emulaattoria.

Testaus on toteutettu niin, että PC-version testaus on suoritettu ensimmäiseksi, ja testaukseen on tehty omat testitapaukset. Ohjelmointivaiheessa eri ohjelmamoduulit eli

aliohjelmat on testattu white box- ja black box -menetelmillä. Tämän jälkeen ohjelmalle on suoritettu vielä järjestelmätestaus.

Tämän jälkeen on testattu sulautetun järjestelmän toiminnallisuuteen toteutettu ohjelma. Testausmenetelmänä ohjelmassa on käytetty moduuli- sekä järjestelmätestausta. Testauksen seuraavassa vaiheessa ohjelmamoduulit on yhdistetty ja ohjelmistolle on toteutettu integraatiotestaus. Lopuksi ohjelmistolle on tehty käyttöliittymä- ja koko järjestelmän järjestelmätestaus.

Testauksessa pyrittiin etsimään systemaattisesti ohjelmistossa esiintyvät virheet ja toteamaan, että ohjelmisto täyttää sille annetut vaatimukset. Testaus toteutettiin testaussuunnitelman ja eri testitapausten pohjalta. Testauksen edetessä V-mallin mukaisesti moduulitasolta järjestelmätestaustasolle löytyi ohjelmasta useitakin virheitä. Testauksen myötä ohjelmasta löytyi myös ns. kuollutta koodia.

Suurin osa virheistä kuitenkin löytyi moduulitestauksen yhteydessä, jossa testauksen kohteena olivat eri aliohjelmat. Eri virhetyypit vaihtelivat toimintalogiikkavirheistä erilaisiin muuttujien alustusongelmiin jne. Esimerkiksi ohjelman tekoälyn vaikeustasoissa ilmeni toimintalogiikkavirheitä, eli toteutus ei vastannut ihan suunnittelua. Testauksessa havaitut virheet korjattiin välittömästi ja testitapauksessa ilmennyt virhe testattiin uudelleen.

Kun virheet saatiin korjatuksi ja moduulitestaus suoritettua, siirryttiin seuraavaan testaustasoon. Suurin osa virheistä kohdistuikin juuri moduulitestausvaiheeseen, joten seuraavilla testaustasoilla ei virheitä juurikaan havaittu tai ne olivat satunnaisia.

6 LOPPUTULOKSET JA ANALYYSIT

Alkuperäisenä ideana tietokonepelaajan tekoälyn toteutuksessa oli käyttää MinMax-algoritmia ja pelipuuideaa. Mutta koska tämä tapa osoittautui aika hankalaksi toteuttaa nimenomaan c-kielellä ja käytetty laitteisto asetti tiettyjä rajoituksia, päädyin toteuttamaan tekoälyn toisenlaisella tavalla.

Tekoälyssä on kuitenkin muutamia toimintamalleja, jotka on otettu MinMax-algoritmi- ja pelipuuideasta. Esimerkiksi kun pelissä halutaan rakentaa kahden suoria, joudutaan tekemään siirtoja ja perumaan niitä, eli pelipuuta käydään läpi ensimmäisellä hakusyvyydellä. Mikäli pelitilanteessa on mahdollista rakentaa kahden suora, saa tietokonepelaaja arvoksi +2. Tämä tilanne vastaa MinMax-algoritmin max-siirtoa. Min-siirtoa ei tekoälyssä ole huomioitu (Ks. luku 5.6 pelipuut ja minmax-algoritmi).

Mikäli ohjelmointikielenä olisi käytetty Javaa tai C++:aa, olisi tekoäly voitu toteuttaa pelipuu ja MinMax-algoritmi-idealla. Vaikeustasot olisi voinut tehdä vaihtelemalla hakusyvyyttä pelipuusta. Tosin tietystä hakusyvyydestä eteenpäin parhaan siirron etsiminen pelipuusta olisi kestänyt 8051:llä ehkä liian pitkään, mikä ei olisi ollut mielekästä.

Peli ei siis ole itseoppiva, vaan ns. itseoppivuus on toteutettu eri vaikeustasoilla. Peli siis vaikeutuu koko ajan pitemmälle pelatessa. Tietokonepelaajan taidot kasvat eri tasoilla, ja näyttää jotakuinkin siltä, että se oppisi aikaisemmista pelatuista eristä.

Tehtävänannon kriteeri ”tietokone voittaa aina tai tulee tasapeli” tuli myös toteutetuksi. Vaikeimmalla tasolla tietokonepelaaja pyrkii ensisijaisesti estämään kaikki pelaajan voittomahdollisuudet ja tämän jälkeen tekemään omalta kannalta hyvän siirron, joka johtaa voittoon. Jätkänshakkipelissähän on tiettyjä strategioita, joilla vastustaja voidaan voittaa, jos niitä ei ennalta tiedä, mutta nämä tietyt strategiat on vaikeimmalla tasolla otettu huomioon.

Lopputulos on tyydyttävä, koska tekoäly toimii hyvin ja peli toimii laitteistossa moitteettomasti. Peli on automaattinen, koska vaikeustasot säätyvät automaattisesti ja pelin loppuessa siirrytään ohjelman alkuun.

7 YHTEENVETO

Työn tarkoituksena oli suunnitella ja toteuttaa ”itseoppiva” jätkänshakkipeliautomaatin ohjelmakoodi c-kielellä 8051-mikrokontrollerille. Laitteistona peliautomaatissa käytettiin valmista testilaitteistoa. Jätkänshakkipeliautomaatti tulee Kajaanin ammattikorkeakoulun esittelykäyttöön. Peliautomaattia käytetään eri tilaisuuksissa, kuten esimerkiksi messuilla, missä esitellään Kajaanin ammattikorkeakoulua sekä sen eri koulutusalavaihtoehtoja. Jätkänshakkipeliautomaatin on tarkoitus esitellä tietotekniikan koulutusohjelman tarjoamia mahdollisuuksia.

Peliä pelataan tietokonepelaajaa vastaan, joka pelien edetessä oppii pelin eri strategiat. Kun täydellinen ”oppiminen” on tapahtunut, lopputuloksena tulee aina pelistä tasapeli tai tietokonepelaaja voittaa. Pelin tekoäly toteutettiin eri vaikeustasoihin. Vaikeustasoa kasvatetaan pelien edetessä. Työssä perehdyttiin myös hieman peliteoriaan työn aiheen tiimoilta.

Tekoälyn ohjelmoiminen oli uutta asiaa, joten asiasta joutui hakemaan jonkin verran esitietoa, ennen kuin sitä pystyi edes suunnittelemaan. Eri toteutusvaihtoehtojen pohdinta tuotti myös hieman päänvaivaa tekoälyn tiimoilta. Osittain ohjelmoinnissa tuli tehtyä turhaakin työtä, kun havaittiin, että MinMax-algoritmia käyttäen tekoälyn ohjelmointi osoittautui aika monimutkaiseksi ja hankalaksi. Kun tarkoituksena oli toteuttaa koodi, joka on yksinkertainen ja kevyt suorittaa 8051:llä.

Itse pelin toiminnallisuus oli helppo toteuttaa. Niinpä aluksi pelistä toteutettiin kahden pelaajan version, johon myöhemmin lisättiin tekoälyosio. Näin toimiessa pelin perustoiminnallisuuden testaus oli helppoa, minkä jälkeen oli hyvä alkaa toteuttamaan peliin tekoälyä.

Myös Graafinen LCD-näyttö oli tuntematon, jonka vuoksi asia vaati jonkin verran perehtymistä asiaan. Muuten laitteisto oli tuttu, joten laitteiston vaatimat ohjelmat olivat helppoa toteuttaa.

Vaikeinta työssä olikin tekoälyn suunnittelu ja ohjelmoiminen. Nykyisessä tekoälymallissa päänvaivaa tuotti vaikeustasojen suunnittelu. Vaikeustasojen välitasot tuli toteuttaa niin, että tietokonepelaaja osaa pelata aina hieman paremmin, mutta ei parhaimmalla mahdollisella

tavalla, jolloin pelaajalle jätetään mahdollisuus voittaa. Työn suoritus onnistui kaiken kaikkiaan hyvin, vaikka työ ei valmistunutkaan laaditussa aikataulussa.

LÄHTEET

1. Ristinollan historia, Luettu 21.2.2006, [WWW-dokumentti], http://www.cs.helsinki.fi/u/hhulkko/tiralab/gomoku.html

2. Avril Styrman, Peliteoria tietojenkäsittelijän työvälineenä, Luettu 10.3.2006, [PDF-dokumentti], www.cs.helsinki.fi/u/myllymak/Teaching/2001/Spring/Seminar/styrman.ps

3. Samuli Siivonen, Othello-pelin evaluointifunktion kehittäminen, Luettu 15.3.2006, [PDF-dokumentti], http://gamics.cs.helsinki.fi/Material/Othello/Samuli_Siivonen_gradu.pdf

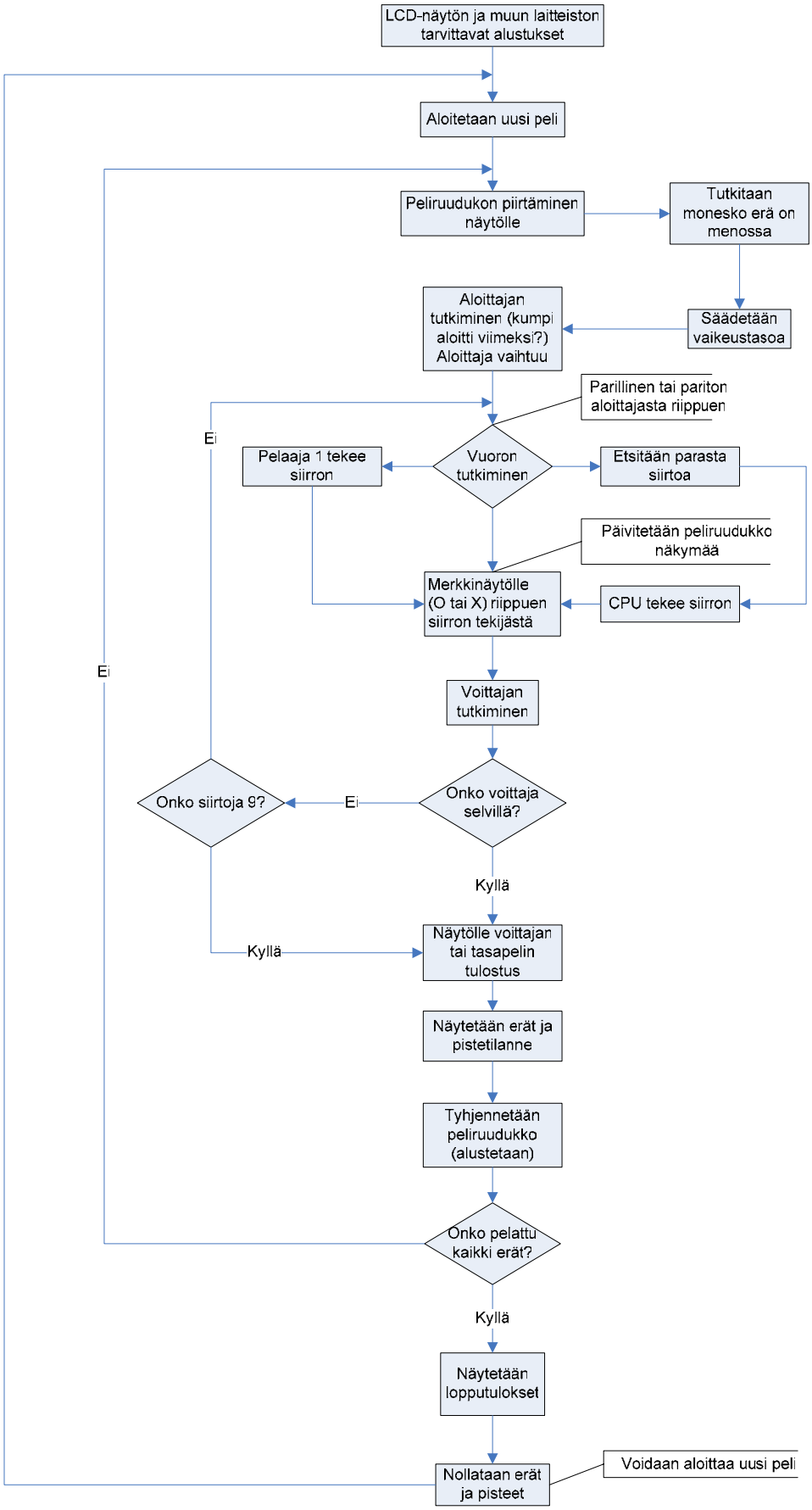
4. Pauli Miettinen, Tekoäly klassisissa strategiapeleissä, tappelupeleissä sekä erinäisissä muissa pelityypeissä, Luettu 10.3.2006, [PDF-dokumenttti], www.cs.helsinki.fi/u/tapasane/Seminaarit/Tietokonepelit/syksy2005/tekoaly_12-14_miettinen_p.pdf

5. Pekka Rantala, Luettu 2.12.2005, [WWW-dokumentti], http://www.tekniikka.oamk.fi/%7Epekkar/omat_kirjat/mcs51.doc

6. Pekka Rantala, Mikrotietokonetekniikka tietokonetekniikka osa 2, TIETOKOTKA, 2001, 86 - 87 s. , ISBN-951-559-258-5

7. Tuomas Kautto, Ohjelmistotestaus ja siinä käytettävät työkalut, luettu 2.9.2006, [HTML-dokumentti], http://www.mit.jyu.fi/opiskelu/seminaarit/ohjelmistotekniikka/testaus/

LIITTEIDEN LUETTELO


LCD-näytön ja muun laitteiston tarvittavat alustukset
Aloitetaan uusi peli
Peliruudukon piirtäminen näytölle
Tutkitaan monesko erä on menossa
Säädetään vaikeustasoa
Aloittajan tutkiminen (kumpi aloitti viimeksi?) Aloittaja vaihtuu
Parillinen tai pariton aloittajasta riippuen
Vuoron tutkiminen
Pelaaja 1 tekee siirron
Etsitään parasta siirtoa
Päivitetään peliruudukko näkymää
CPU tekee siirron
Merkkinäytölle (O tai X) riippuen siirron tekijästä
Voittajan tutkiminen
Onko voittaja selvillä?
Ei
Onko siirtoja 9?
Ei
Kyllä
Kyllä
Näytölle voittajan tai tasapelin tulostus
Näytetään erät ja pistetilanne
Tyhjennetään peliruudukko (alustetaan)
Onko pelattu kaikki erät?
Ei
Kyllä
Näytetään lopputulokset
Nollataan erät ja pisteet
Voidaan aloittaa uusi peli