Valtteri Karhu

Logiikkaohjelmoinnin verkkosivusto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

1.5.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Valtteri Karhu Logiikkaohjelmoinnin verkkosivusto 22 sivua + 1 liite 1.5.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Sähkö- ja automaatiotekniikka
Ammatillinen pääaine	Automaatiotekniikka
Ohjaajat	Lehtori Raisa Vartia

Opinnäytetyön aiheena on logiikkaohjelmoinnin verkkosivusto. Projektin tarkoituksena oli rakentaa sivusto, jossa voi harjoitella function block -diagrammin ohjelmointia.

Sivut on tarkoitettu koulutuskäyttöön automaatiotekniikkaa opiskeleville. Samalla se tarjoaa opettajille hyödyllisen opetusvälineen. Sivusto soveltuu myös logiikkapiirien demonstroimiseen. Sivusto on pääosin ohjelmoitu javascript-ohjelmointikielellä.

Tämä on visuaalisesti selkeä ja helppokäyttöinen sovellus, joka mahdollistaa FBD-ohjelmoinnin perusteiden omaksumisen. Projektin suuruuden takia tämä ei ole vielä kaupallisessa muodossa. Sivustoa jatketaan tulevaisuudessa.

Logiikkaympäristö toimii jo hyvin, ja sitä on helppo kehittää. Sivuston rakenne mahdollistaa uusien ominaisuuksien lisättävyyden. Uudenlaisena menetelmänä se täydentää tulevaisuudessa automaatiotekniikan opetusta ja helpottaa logiikkaohjelmoinnin harjoittelemista.

Avainsanat	Ohjelmointi, FBD, automaatiotekniikka, sivusto

Author Title Number of Pages Date	Valtteri Karhu Logic Programming Website 22 pages + 1 appendix 1 May 2018
Degree	Bachelor of Engineering
Degree Programme	Electrical and Automation Engineering
Professional Major	Automation Engineering
Instructors	Raisa Vartia, Senior Lecturer

This thesis is about a website, where you can study PLC-programming. The goal for this project was to create an environment where you can build your own custom code with function block diagram method.

The website is meant to be used as a learning tool for students. It also provides a new way for teachers to teach automation engineering. Furthermore, it is a useful tool to test your own creations in an easy to access sandbox. The website was built with Javascript programming language.

This project produced a clear and easy to use environment to build a custom FBD code. Because of the sheer scale of the project, it is not yet in a marketable format. This software will be developed further.

The website is also easy to expand. The way it is built, makes it easy to add new functionalities such as the other standardized PLC programming languages. Currently the teaching methods for PLC programming are scarce and difficult to maintain by a teacher. This new way of teaching will make it easier for students to understand how logic controllers work.

Keywords	Programming, FBD, Automation engineering, website

Sisällys

Liite 1. Tilastotietoa projektista

Lyhenteet

CSS	Cascading Style Sheets. Ohjelmointikieli, jolla muutetaan verkkosivujen visuaalista puolta.
Drag'n'drop	Drag and drop. Sovelluksen käyttöön viittaava ilmaus, jossa toimintona on esineen siirto hiiren vasenta painiketta pohjassa painamalla.
FBD	Function Block Diagram. Automaatiossa käytetty logiikan ohjelmointikieli.
Framerate	Ohjelman suoritusnopeus. Logiikkaympäristö on simuloitu peruslogiikkaohjaimen suoritusnopeuden mukaan.
GVL	Global Variable List. Logiikkaohjelmoinnissa viittaus muuttujiin, joihin pääsee käsiksi joka paikasta ohjelmaa.
HTML	Hyper Text Markup Language. Verkkosivujen standardi ohjelmointikieli.
IO	Input/Output. Logiikkaohjelmoinnissa käytetty nimike sisään- ja ulostuloille logiikkakomponenteilta.
Javascript	Ohjelmointikieli, joka tuo lisäominaisuuksia HTML -pohjaisiin verkkosivuihin.
MYSQL	Tietokanta, johon serverikone tallentaa esimerkiksi käyttäjätiedot.
Objekti	Nimitys, joka viittaa verkkosivuilla piirrettyyn asiaan. Tämä voi olla Function block -diagrammin mukainen function block tai johdonpätkä, joka kulkee näiden palikoiden välillä.
OOP	Object Oriented Programming on ohjelmointimenetelmä, jolla jaetaan ohjelmointikieltä oliopohjaiseksi.
PHP	Hypertext Preprocessor. Verkkosivujen serveripuolen ohjelmointikieli.

1 Johdanto

Opinnäytetyön tarkoituksena oli rakentaa verkkosivusto, jota voidaan käyttää automaatiotekniikan opiskelijoiden logiikkaohjelmoinnin harjoitteluun. Tällä verkkosivustolla on mahdollista harjoitella FBD-ohjelmointikieltä verkkosivulle suunnitellussa ohjelmointiympäristössä. Sain idean opinnäytetyöni aiheeseen opiskellessani ohjelmointia sivuaineena Etelä-Koreassa Ajou Universityssä. Erään kurssin aiheena oli rakentaa ryhmätyönä sovellus. En kuitenkaan halunnut toteuttaa tätä ideaani kyseisellä kurssilla, koska kurssikaverini eivät opiskelleet automaatiotekniikkaa.

Ammattikorkeakoulussa opiskelimme logiikkaohjelmointia ainoastaan laboratoriotunneilla ryhmätöinä fyysisten logiikoiden parissa. Logiikkaohjelmointisimulaattori tukisi hyvin opiskelua, sillä se edellyttäisi jokaisen opiskelijan aktiivista työskentelyä. Ryhmätöissä tämä ei aina välttämättä toteutunut. Opiskelu olisi hyvä aloittaa verkkoharjoitusten avulla, ja vasta kun ohjelmoinnista on jo pieni ymmärrys, siirryttäisiin fyysisiin komponentteihin laboratoriotiloissa.

Logiikkaohjelmoinnin harjoittelu antaa myös hyvät lähtökohdat työelämään. Moni nykyisistä automaatioalan ammateista liittyy logiikkaohjelmointiin ja näiden ohjelmien muokkaamiseen. Hyvä harjoitusympäristö valmentaisi opiskelijat juuri tällaisiin tehtäviin.

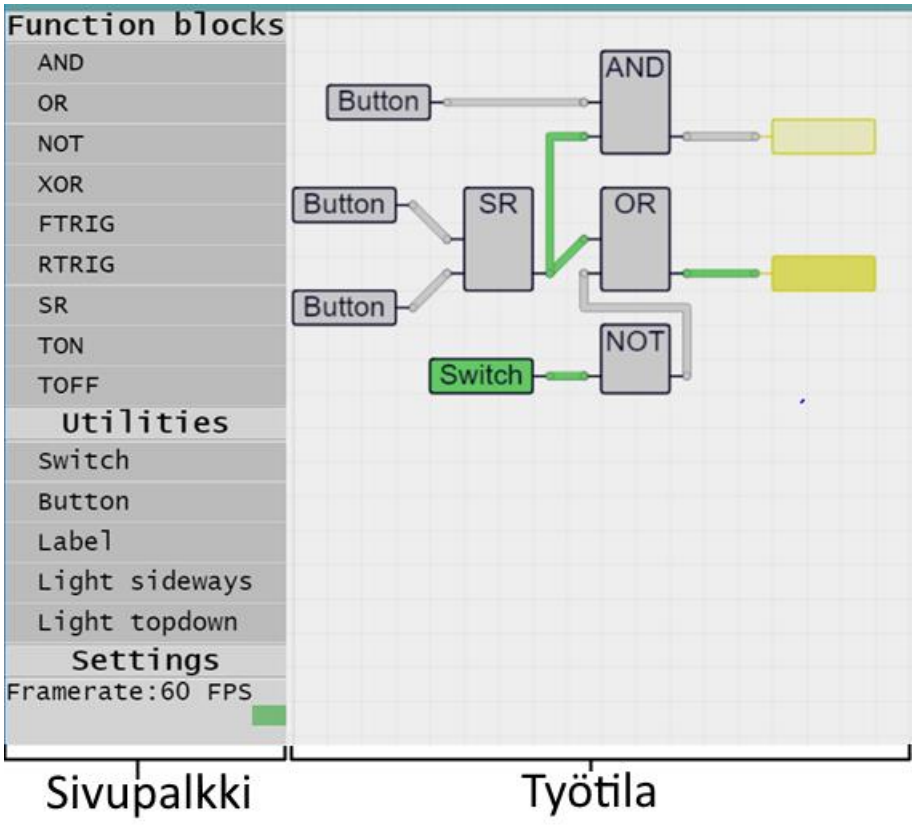
Pyrkimyksenä oli luoda helppokäyttöinen ja selkeä ohjelmointiympäristö opiskelijoille, joilla ei ole vielä kokemusta logiikkaohjelmoinnista. Tällainen ympäristö olisi myös hyvä työelämässä demonstroimiseen sekä logiikkaohjelmien testaamiseen. Verkkosivusto on myös opettajille hyvä opetustyökalu.

Opetusta on alettu digitalisoimaan. Opiskelumateriaalin digitalisoinnista on huomattavasti hyötyä. Tämä on todettu jo monella muullakin alalla [1; 2; 3]. Harjoitusten digitalisointi helpottaa myös aikuisopiskelua. Välttämättä työn ohessa ei aina ole mahdollista päästä luennoille. Digitaalisessa ympäristössä opiskelija voi harjoitella omaan tahtiinsa omalla aikataulullaan.

2 Verkkosivuston ulkoasu

Kuvassa 1 on oikealla työtila. Tätä ympäristöä käytetään ohjelmien rakentamiseen. Työtilassa voi palikat yhdistää johdoilla ja näin toteuttaa function block -diagrammin mukaisesti erilaisia logiikkasovelluksia.

Kuvan 1 vasemmalla puolella on lista objekteista, jotka voi viedä työtilalle drag'n'drop-menetelmällä. Tässä palkissa on myös Settings-välilehti. Framerate-asetuksella voi vaihtaa logiikkapiirin suoritusnopeutta. Tämän lisäyksen avulla on helpompi simuloida oikeita logiikkakomponentteja, ja se myös havainnollistaa logiikan toimivuutta step by step -menetelmällä.



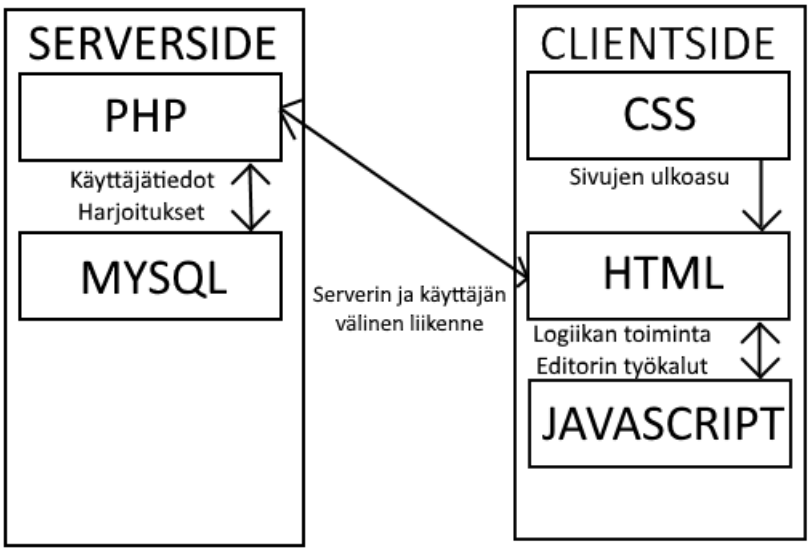
Kuva 1. Verkkosivusta kokonaiskuva

3 Sivuston toiminnallisuus

3.1 Tekninen toteutus

Kuvassa 2 on esitetty projektin eri komponenttien välinen kommunikaatio. PHP rakentaa HTML-, Javascript- ja CSS-kokonaisuuden, joka lähetetään käyttäjälle. Tämä kokonaisuus riippuu siitä, millä sivulla käyttäjä on sillä hetkellä.

PHP [4] on serveripuolen koodi, jossa on käyttäjän sisäänkirjautumisfunktiot. PHP:n välityksellä myös yhdistetään käyttäjä MYSQL-puolelle.



Kuva 2. Tietoliikenne eri komponenttien välillä

Suurin osa verkkosivujen toiminnallisuudesta on Javascript-ohjelmointikielen alla [5]. Tähän on ohjelmoitu logiikkapalikoiden piirto, toiminnallisuus sekä verkkosivuille ohjelmoidun editorin eri työkalut.

HTML [6] on jäänyt tämän verkkosivuston aiheen takia vähäiseksi. Normaalisti verkkosivut koostuvat melkein kokonaan HTML-koodista ja CSS-tyylittelystä. Koska tämä projekti vaati elementtejä, joita on vaikea toteuttaa internetprotokollan myötä, piti ohjelman olla Javascriptillä ohjelmoitu. Työtila on toteutettu kokonaan HTML <canvas> -elementtiä käyttäen ja tämän piirto sekä toiminnallisuus ovat Javascriptin puolella. Kuvassa näkyvä sivupalkki on ohjelmoitu HTML-elementtejä käyttäen.
CSS [7] on HTML:n tyylittelyä varten. Tällä muokataan verkkosivuston ulkoasua.

PHP rakentaa käyttäjälle HTML-näkymän, johon on tuotu vain kyseiselle välisivulle vaaditut komponentit.

MYSQL on varasto käyttäjän tiedoille sekä harjoituksille. Tulevaisuudessa käyttäjä voi myös tallentaa rakentamansa kokonaisuudet tänne. Työtila on reaaliaikainen. Tämä auttaa ohjelman visualisoinnissa ohjelmoinnin aikana. Se nopeuttaa myös ohjelmoitujen kokonaisuuksien testaamista.

3.2 Olio-ohjelmointi

Sivusto on suunniteltu olio-ohjelmointimenetelmää käyttäen [8]. Jokaiselle function blockille ja johdon pätkälle on oma olionsa. Tähän olioon on tallennettu erilaisia funktioita, mitä kyseinen olio tarvitsee. Olio-ohjelmointi on hyvä ratkaisu tähän projektiin, sillä se vähentää vaadittua ohjelmointimäärää huomattavasti.

Kuvan 3 mukaisesti fbd.Object-oliolla on useita eri funktioita, jota olio voi käyttää. UpdateData-funktion avulla voimme määrittää eri ominaisuuksia function blockista. Tällä voi esimerkiksi määrittää, minkälaista funktiota palikka käyttää ja mikä teksti piirretään palikan päälle.

Oliot ovat myös tallennettu listaan, joka sisältää kaikki työtilassa olevat objektit. Tätä listaa hyväksikäyttämällä voimme esimerkiksi suorittaa jokaisen funktiopalikan päivitystarkastuksen simuloidun frameraten mukaan.

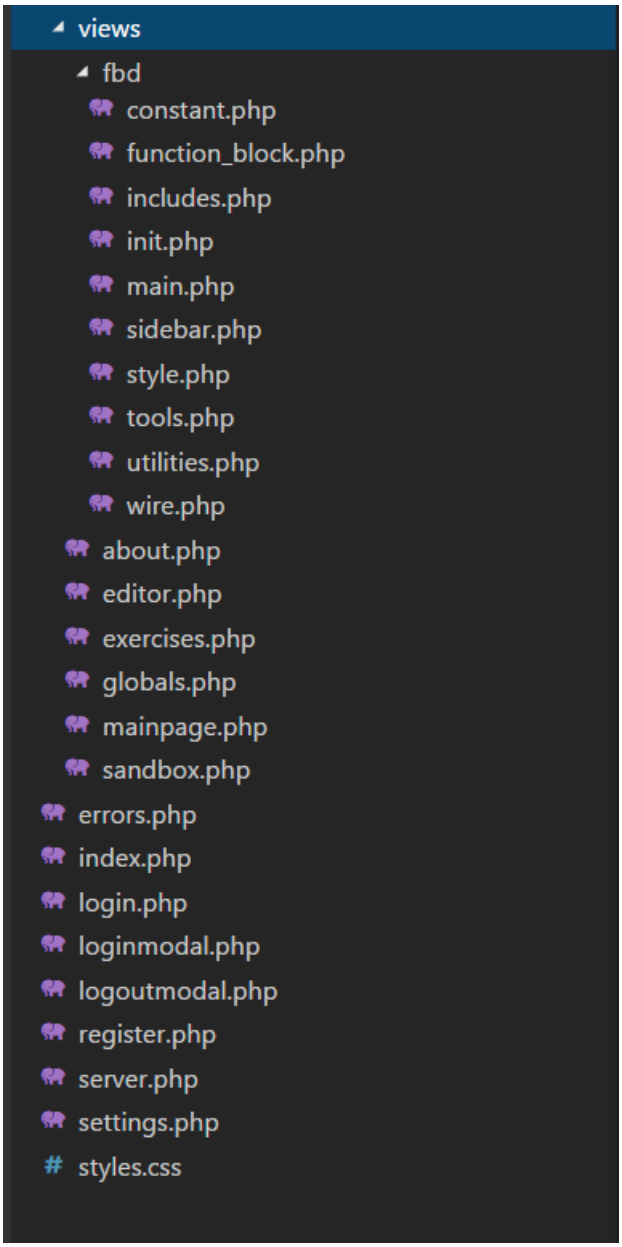
Jokaisella oliolla on myös draw-funktio. Tämä funktio piirtää kaikki ruudulla olevat objektit näytölle käyttäen Javascriptin requestAnimationFrame()-funktiota. Tämä funktio on optimoitu kaikille koneille ja sen suoritusnopeus hidastuu, jos koneella ei ole tehoa verkkosivujen pyörittämiseen.

```
fbd.Object = function(canv, data) {
    var c = canv
    this.id = data.id;
    this.selected = false;
    this.outputVal = data.outputVal || false;
    this.updateData = function(data) {…
    }
    this.updateData(data);
    var txtSpacing = 10;
    this.draw = function(data) {…
    }
    this.center = function() {…
    }
    this.getSize = function() {…
    }
    this.collides = function(data) {…
    }
    this.delete = function() {…
    }
    this.tick = function() {…
    }
    this.setOutput = function(value) {…
    }
    this.click = function() {
    }
}
```

Kuva 3. Function block -olio

3.3 Sivuston rakenne

Sivusto on tarkoituksella rakennettu useaan eri kappaleeseen (Kuva 4). Tämä mahdollistaa uusien ominaisuuksien lisäämisen helposti. Uuden ohjelmointikielen lisäys tässä tapauksessa tapahtuisi siten, että views-kansioon lisättäisiin ohjelmointikielelle oma kansionsa. Server.php ohjaa sen jälkeen, mitä näistä kansioista käytetään ohjelman juoksemiseen.



Kuva 4. Koodipuun rakenne

Views-kansion alla on fbd-kansio. Se sisältää includes.php-tiedoston (Esimerkkikoodi 1), joka on kokonaisuudessaan vain pieni pätkä koodia. Tämä lisää FBD-koodiin kaikki tarvittavat ominaisuudet oikeassa järjestyksessä. Tällä tavalla saadaan koodi jaettua ja pidettynä kokonaisuus siistinä. Samanlainen koodirakenne on myös muualla käytössä. Pääsivulla includen avulla lisätään haluttu sivu views-välilehden alta.

```
<?php
    include(__DIR__."/../globals.php");
    include("init.php");
    include("style.php");
    include("function_block.php");
    include("wire.php");
    include("constant.php");
    include("main.php");
    include("tools.php");
    include("utilities.php");
    include("sidebar.php");
?>
```

Esimerkkikoodi 1. FBD includes toiminta

3.4 Työkalujen toiminnallisuus

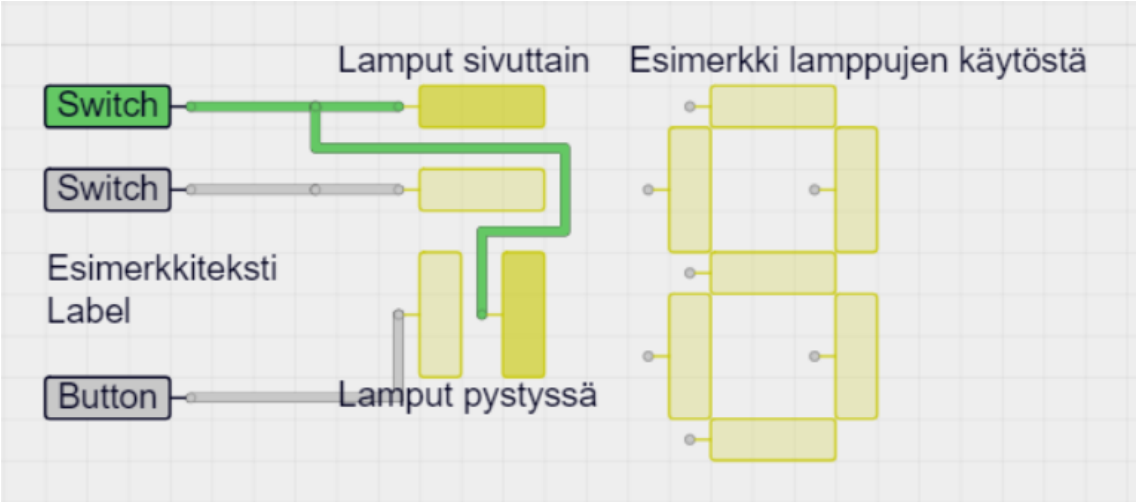
Tässä luvussa selitetään, miten erilaisia työkaluja käytetään verkkosivulla. Nämä on suunniteltu siten, että niitä olisi helppo käyttää ja erilaisia ominaisuuksia ei tarvitsisi etsiä editorista liiallisten työkalupalkkien alta. Selityksessä käytetään myös sanoja sivupalkki ja työtila. Nämä viittaavat kuvaan 1.

3.4.1 Logiikkaobjektien tuonti

Erilaisia logiikkakomponentteja voi luoda sivun vasemmalla puolella olevasta sivupalkista. Tämä toimii drag'n'drop-tyylillä, eli käyttäjän täytyy painaa hiiren ensimmäinen painike pohjaan valitsemansa komponentin yläpuolella ja sen jälkeen raahata hiiri työtilaan, jossa komponentti sijoitetaan käyttäjän vapauttaessa hiiren.

3.4.2 Lisäominaisuudet

Editorissa on myös lukuisia lisäominaisuuksia, jotka auttavat logiikan ymmärrystä ja visualisointia. Näitä ovat esimerkkinä painonapit, Label sekä erilaiset lamput. Nämä mahdollistavat työtilassa omien harjoitusten luonnin sekä helpottavat visualisoimaan käyttäjän rakentamaa koodia. Näitä nappeja on myös kahdenlaisia. Button-niminen nappi on aktiivinen vain silloin kun nappia pidetään pohjassa. Switch-niminen nappi menee päälle nappia painettaessa ja pois päältä vasta sitten, kun nappia painetaan toisen kerran. Kuvassa 5 on vielä näytetty nämä ominaisuudet.



Kuva 5. Utilities-välilehden objektit

3.4.3 Työkalujen vaihto

Kuvassa 6 näkyy, miten työkalujen vaihto on toteutettu ohjelmallisesti. Changetool-funktion alla otetaan valitun työkalun funktiot tai jos niitä ei ole olemassa, niin ne yli kirjoitetaan tyhjillä funktioilla. Joissakin näissä työkaluissa ei tarvita kaikkia mahdollisia event-funktioita, joten tämä funktion ylikirjoitustyyli on helpoin ratkaisu koodin vähentämiseksi.

```
<script>
    function changetool(mode, data) {
        if(_tools[toolmode].exit) {
            _tools[toolmode].exit();
        }
        tooltip = "";
        toolmode = mode;
        tooldata = data || {};
        toolinit = false;
        tool_mousedown = _tools[mode].mousedown || function() {};
        tool_mouseup = _tools[mode].mouseup || function() {};
        tool_rightclick = _tools[mode].rightclick || function() {};
        tool_middleclick = _tools[mode].middleclick || function() {};
        tool_mousemove = _tools[mode].mousemove || function() {};
        tool_render = _tools[mode].render || _tools[""].render;
        tool_keyevent = _tools[mode].keyevent || function(key) {};
        tool_keypress = _tools[mode].keypress || function(key) {};
        _tools[mode].init();
        getHoveredObject();
    }
    var _tools = {};
    _tools[""] = {···
    }
    _tools["dragndrop"] = {···
    }
    _tools["wiring"] = {···
    }
    _tools["buttonpress"] = {···
    }
    _tools["select"] = {···
    }
    _tools["selected_objects"] = {···
    }
    _tools["label_input"] = {···
    }
    _tools["paste"] = {···
    }
    _tools["TxtBox"] = {···
```

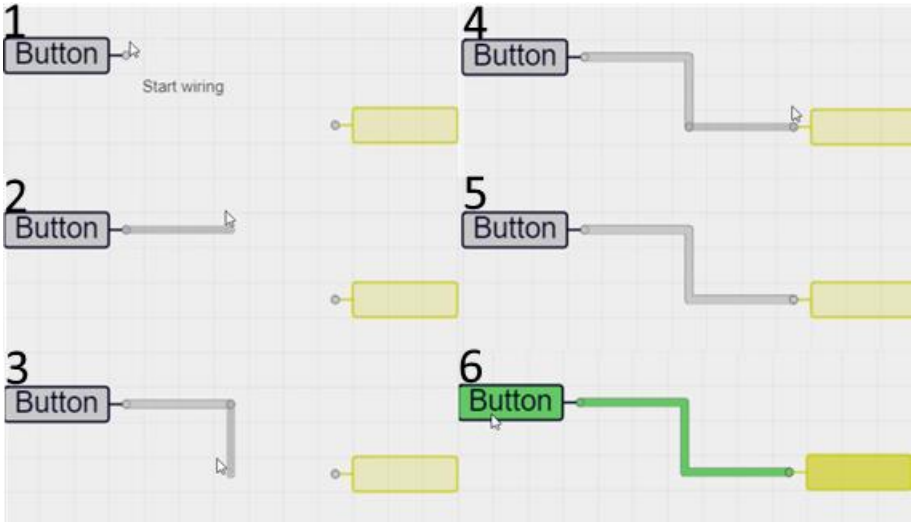
Kuva 6. Työkalujen vaihto

3.4.4 Johdotus

Johdotus vaatii, että työtilaan on jo tuotu komponentti, jossa on johdotuspaikka sisäänrakennettuna. Kaikissa logiikkapalikoissa on ainakin yksi tulo- sekä lähtöjohto.

Johdotuksen voi aloittaa kuvan 7 esimerkin mukaisesti siten, että hiiren vasenta painiketta painetaan silloin kun hiiri on johdon päällä (1). Tämä aloittaa johdotustyökalun käytön. Seuraavaksi voi siirtää hiirtä haluttuun paikkaan (2). Läpinäkyvä johto piirretään aloituspaikasta hiiren osoittamaan paikkaan. Klikkaamalla uutta kohtaa saa johdolle uuden

kääntymispisteen (3). Tämän jälkeen voidaan hiiri taas siirtää uuteen kohtaan, johon halutaan johdon jatkuvan. Johdottamisen voi lopettaa joko painamalla toista johtoa tai hiiren oikeaa painiketta. Mikäli toista johtoa painetaan johdottamisen aikana niin nämä kaksi johtoa yhdistyvät toisiinsa (4; 5). Kuvan 7 esimerkin mukaan painonappi on yhdistetty oikealla olevaan lamppuun. Tätä painonappia painettaessa syttyy lamppu päälle (6). Johto näkyy vihreällä värillä silloin, kun se on päällä.



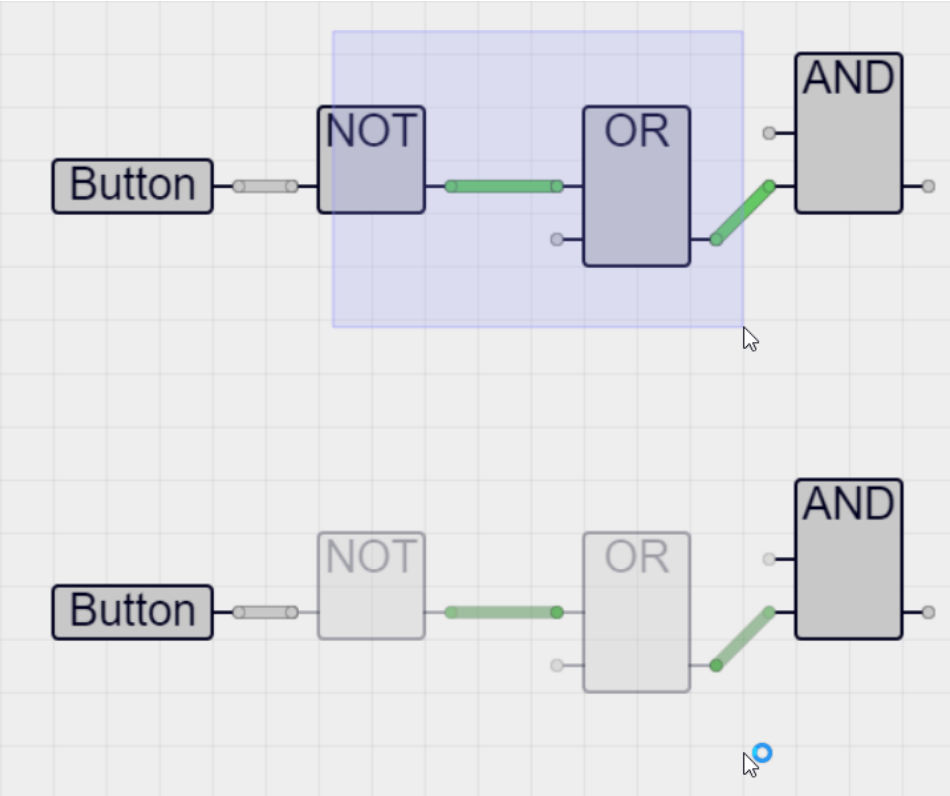
Kuva 7. Johdottamisen vaiheet painonapilta lampulle

3.4.5 Valinta

Työtilassa objektien valinta toimii samalla lailla, kuin monessa muussakin editorissa. Valintalaatikon voi piirtää kuvan 8 mukaisesti siten, että painetaan hiiren vasenta painiketta pohjaan missä vain kohdassa, jossa ei ole mitään objektia. Tämän jälkeen käyttäjän täytyy liikuttaa hiirtä. Valintalaatikko piirtyy kohdasta, josta käyttäjä aloitti hiiren painikkeen painamisen kohtaan, missä hiiri on parhaillaan. Jos käyttäjä vapauttaa hiiren painikkeen tässä tilanteessa, laatikon sisällä olevat kappaleet valitaan ja ne piirretään hiukan läpinäkyvinä.

Valituille kappaleille voi tehdä erilaisia asioita. Näppäimistön Delete-painike poistaa valitut objektit. Kappaleita voi siirtää drag’n’drop-tyylillä. Kappaleet voidaan kopioida

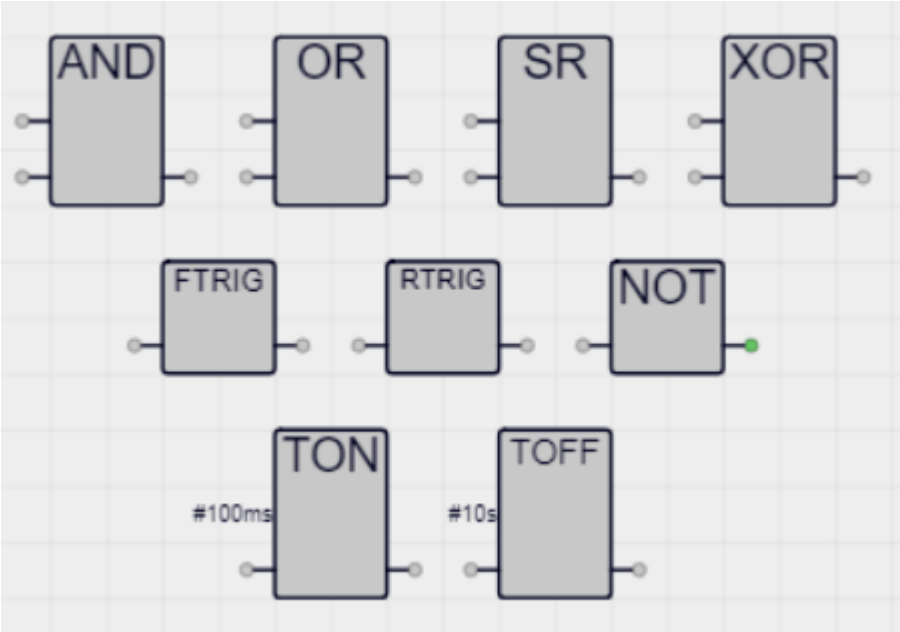
painamalla ctrl- ja c-painikkeita samanaikaisesti. Kopioidut kappaleet voidaan syöttää työskentelyalueelle takaisin painamalla ctrl- ja v-painiketta, jonka jälkeen hiirtä liikuttamalla voidaan valita näille uusi positio.



Kuva 8. Valintatyökalun toiminnallisuus

4 Logiikkaobjektit

Jokaisessa logiikkaobjektissa on vasemmalla puolella tulo ja oikealla puolella lähtö. Logiikkakomponentti ohjaa lähdössä olevaa johtoa input-johtojen mukaan. Logiikkakomponentissa on oma koodinsa, jolla se ohjaa lähtöjohtoaan eri tavoilla. Kuvassa 9 on nykyiset ohjelmoidut function blockit [9].



Kuva 9. Nykyiset logiikkaobjektit

AND-palikalla (Esimerkkikoodi 2) on kaksi sisääntuloa. Palikka ohjaa ulostulon silloin päälle, kun molemmat input johdot ovat päällä. Tämän ulostulon ohjaus tapahtuu seuraavalla ohjelmakierrolla.

```
var _on = true;
for(var i=0; i<this.inputs.length; i++) {
    if(!this.inputs[i].active()) {
        _on = false;
    }
};
this.setOutput(_on);
```

Esimerkkikoodi 2. AND-blockin logiikka

OR-palikalla (Esimerkkikoodi 3) on kaksi sisääntuloa. Palikka ohjaa lähdön päälle silloin, kun toinen tai molemmat sisääntuloista on päällä.

```
var _on = false;
for(var i=0; i<this.inputs.length; i++) {
    if(this.inputs[i].active()) {
        _on = true;
    }
};
this.setOutput(_on);
```

Esimerkkikoodi 3. OR-blockin logiikka

SR (Esimerkkikoodi 4), eli Set Reset -palikalla on kaksi sisääntuloa. Ylempi näistä on Set ja alempi Reset. Ylempi sisääntulo laittaa lähdön päälle ja alempi nollaa tämän.

```
if(this.inputs[0].active()) {
    this.sr_on = true;
}else if(this.inputs[1].active()) {
    this.sr_on = false;
};
this.setOutput(this.sr_on);
```

Esimerkkikoodi 4. Set Reset -palikan logiikka

XOR (Esimerkkikoodi 5) on päällä vain silloin, kun jompikumpi sisään tulevista johdoista on päällä. Jos molemmat tai ei kumpikaan ole päällä, niin silloin lähtö on pois päältä.

```
var _on = false;
if(this.inputs[0].active()!=this.inputs[1].active()) {
    _on = true
};
this.setOutput(_on);
```

Esimerkkikoodi 5. SR-palikan logiikka

FTRIG (Esimerkkikoodi 6) eli falling trigger toimii siten, että input-johdon mennessä päältä pois käy output yhden ohjelmakierron päällä. RTRIG eli rising trigger aktivoituu yhdeksi ohjelmakierroksi silloin, kun sisään tuleva johto menee päälle.

```
if(!this.inputs[0].active() && this.initialized) {
    if(!this.output_set) {
        this.setOutput(true);
    }else{
        this.setOutput(false);
    }
    this.output_set = true;
}else{
    this.output_set = false;
    this.setOutput(false);
}
this.initialized = true;
```

Esimerkkikoodi 6. FTRIG-palikan logiikka

Not-palikka (Esimerkkikoodi 7) invertoi tulon. Eli jos inputiin syöttää ON-käskyn, menee output-johto OFF-tilaan.

```
this.setOutput(!this.inputs[0].active());
```

Esimerkkikoodi 7. NOT-palikan logiikka

TON (Esimerkkikoodi 8) eli Timer ON delay laittaa outputin päälle sitten, kun input-johto on ollut määritetyn ajan päällä. TOFF eli Timer OFF delay laittaa outputin päälle, kun input-johto on aktiivinen. Se myös pitää outputia niin pitkään päällä, kunnes määritetty aika umpeutuu.

Molemmissa objekteissa on kaksi sisääntuloa. Toinen näistä on aikamääre, jota voi muuttaa sitä klikkaamalla ja kirjoittamalla uuden ajan.

```
var time = new Date();
if(this.inputs[1].active()) {
  this.started = true;
  if(this.time + this.inputs[0].value > time.valueOf() &&
this.timeSet) {
    this.setOutput(false);
  }else if(this.timeSet) {
    this.setOutput(true);
  }
}else{
  this.timeSet = false;
  this.started = false;
  this.setOutput(false);
 }
 if(this.started && this.inputs[1].active() && !this.timeSet) {
   this.timeSet = true;
   this.time = time.valueOf();
 }
```

Esimerkkikoodi 8. TON-palikan logiikka

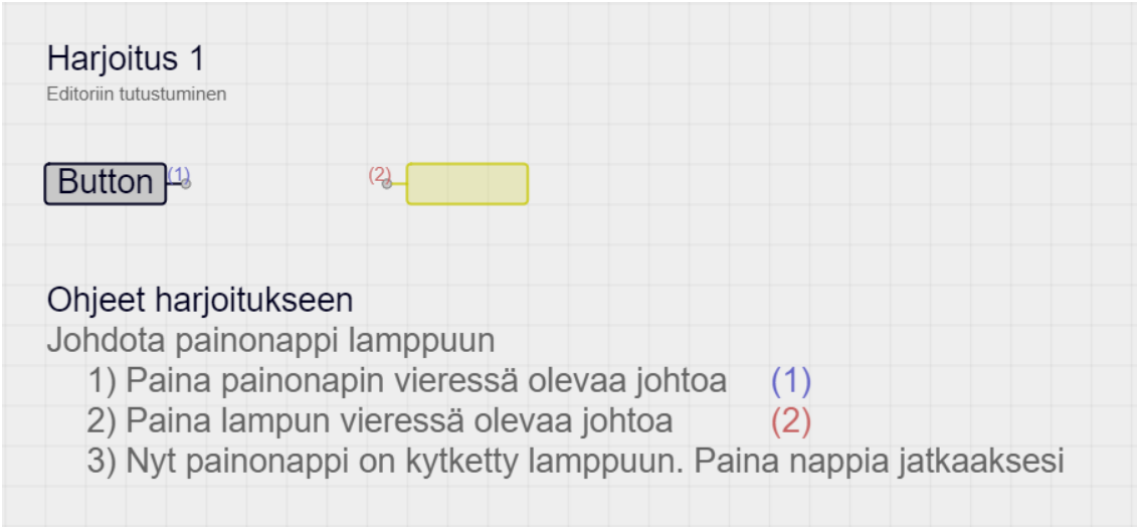
5 Esimerkkiharjoituksia

5.1 Harjoitusten tavoite

Harjoitusten päätavoite on saada opiskelijat ajattelemaan automaatiossa vastaan tulevia haasteita ohjelmoijan silmin. Ohjelmointikielten käyttämiseen vaaditaan yleiskuva niistä sekä siitä, miten erilaiset komponentit toimivat loogiselta kannalta. Harjoitusten tulee siis olla sellaisia, että ne opettavat ihmisiä ajattelemaan, miten logiikkaohjaimet toimivat. Vasta tämän ymmärryksen jälkeen voi työelämässä ohjelmoida sujuvasti logiikoita. Haastavin asia opettamisessa on mielenkiinnon ylläpito luennoilla. Harjoituksissa tulee olla tarpeeksi vaihtelua ja haastetta opiskelumotivaation ylläpitämiseen.

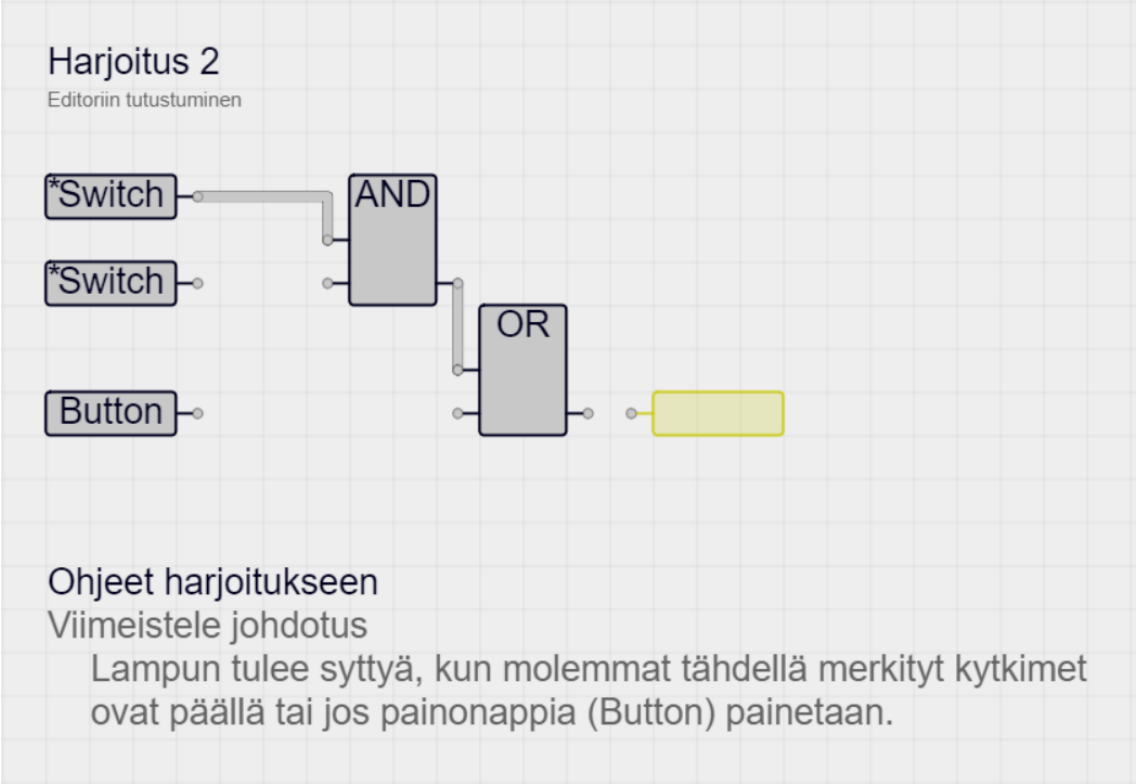
5.2 Alkeet

Aloittaisin opetuksen kuvan 10 kaltaisella helpolla harjoituksella. Se valmentaa opiskelijat ohjelmointiympäristöön vaatimatta opiskelijoilta vielä logiikkaohjelmoinnin kokemusta. Harjoituksen periaatteena on tutustuttaa opiskelija function block -diagrammin tärkeimpään ominaisuuteen, eli palikoiden toisiinsa johdottamiseen. Alkuvaiheessa harjoituksessa tulee myös olla selkeät ohjeet näiden ratkaisujen löytämiseen.



Kuva 10. Ensimmäinen harjoitus

Seuraavan harjoituksen tulisi olla jo hiukan haastava. Tässä vaiheessa ei välttämättä tarvitse kertoa vielä function blockien toiminnallisuudesta, vaan opetuksen voi myös toteuttaa ohjaamalla opiskelija tällaiseen helpon piirin toteutukseen. Kuvan 11 mukaisessa harjoituksessa johdotettavat asiat on sijoitettu samalla tavalla kuin ensimmäisessä harjoituksessa, eli johdot tulee kytkeä vain vasemmalta oikealle. Harjoituksen jälkeen tai jos opiskelijalla on vaikeuksia suoriutua siitä, olisi hyvä kerrata lyhyesti, miten kukin harjoituksessa oleva kappale toimii. Kuvan 11 harjoituksessa avattaisiin AND- ja OR-palikoiden toiminnallisuutta.



Kuva 11. Toinen harjoitus

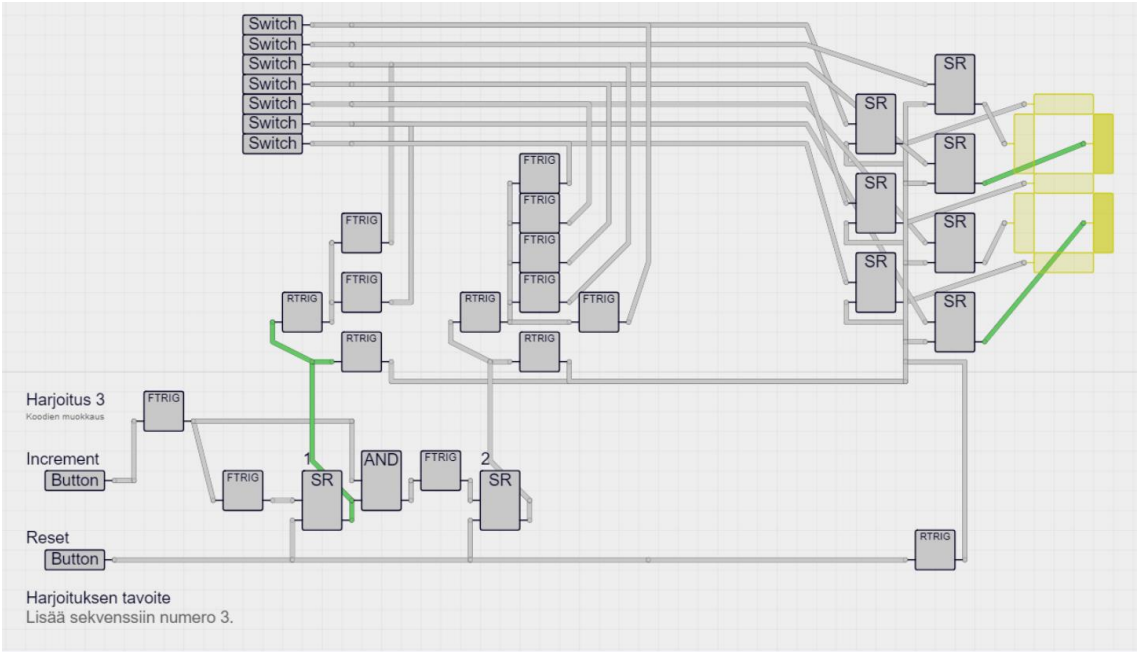
Ensimmäisten harjoitusten aikana olisi myös hyvä pitää sivupalkki siistinä. Eli sivupalkista voisi piilottaa ne kappaleet, joita ei vielä ole käyty läpi harjoituksissa.

5.3 Haastavammat harjoitukset

Haastavimmissa harjoituksissa laittaisin opiskelijan tutustumaan valmiina olevaan ohjelmaan. Näissä harjoituksissa opiskelijan tulisi ensin tutustua siihen, miten alkuperäinen koodi toimii ja miten vaadittu muutos ohjelman toiminnallisuuteen saataisiin ohjelmoitua.

Tämänkaltaiset harjoitukset valmentavat opiskelijoita työelämään, jossa suurin osa ohjelmoinnista on muutosten tekoa.

Kuvan 12 harjoituksessa on tarkoituksena perehtyä siihen, miten nykyinen logiikka toimii ja lisätä siihen vaadittu ominaisuus. Tässä harjoituksessa Increment-painonappia painettaessa oikealla reunassa oleva numero kasvaa yhdellä. Tämä laskuri on ohjelmoitu vain numeroille 1 ja 2. Opiskelijan tarkoituksena on lisätä numero 3 tämän laskurin loppuun. Opiskelijan täytyy siis kopioida SR-piirin (2) mukaisesti AND-, FTRIG- ja SR-palikka ja johdottaa ne samalla tavalla kuin SR-piirissä (2). Tämän jälkeen opiskelijan tulee vetää johdotus, joilla saadaan numero 3 näkyviin kyseisille lampuille.



Kuva 12. Haastava harjoitus

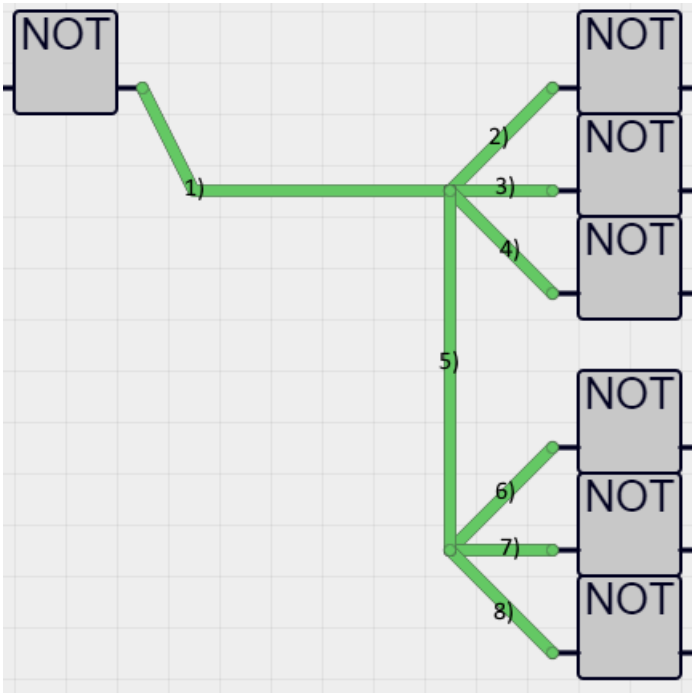
6 Esimerkkejä ohjelmoinnin haasteista

6.1 Johtojen yhdistäminen toisiinsa

6.1.1 Ongelman selitys

Ohjelmaa luodessa ilmeni ongelma, miten johdot saataisiin yhdistettyä toisiinsa, jos niitä olisi useampi kuin kaksi. Editori on kuitenkin reaaliaikainen, eli käyttäjän tekemien

muutosten jälkeen tulee myös muutoksen seuraukset ottaa huomioon. Kuvan esimerkissä on kahdeksan kappaletta johtoja kytketty toisiinsa. Ohjelmallisesti oli haasteellista toteuttaa se, että kun joku johdoista on päällä, niin kaikki muutkin näihin johtoihin kytketyt johdot tulevat päälle. Kuvan 13 tilanteessa myös pitää laittaa johdot 6, 7 ja 8 pois päältä, jos johto 5 poistetaan välistä.



Kuva 13. Yhdistetyt johdot. Numeroilla eroteltu johdot toisistaan (Kuvassa 8 kpl johtoja)

6.1.2 Ongelman ratkaisu

Ongelmaa korjatessa johdotuslogiikka oli ohjelmoitava kolmella eli tavalla, kunnes viimein päädyttiin lopputulokseen, jossa ei tarvitse miettiä logiikan toiminnallisuuden rajatilanteita.

Jokainen johto muistaa, mihin johtoihin se on kiinnitetty. Esimerkiksi kuvassa 13 johto 8 on kiinnitetty johtoihin numero 6, 7 ja 5. Jos johto 8 poistetaan, tämä tieto päivitetään johdoille, jotka ovat kiinni johto 8:ssa.

Jokaisella johdolla on 2 tilamuuttujaa: johdon oma ON-tila ja johdon ulkoinen ON-tila. Johdon muuttaessa sisäistä tilaansa, täytyy myös tähän johtosarjaan olevien johtojen päivittyä. Esimerkkikoodi 9 on ratkaisu tähän ongelmaan.

```
this.updateStatus = function(value) {  ← Johdon tilamuutos
  var loopedArray = [this];
  var foundConnections = 1;
  var wireArray = [this];
  this.on = value;←Johdon sisäinen ON tila
  var state = this.on;
  var offset = 0;
  while(foundConnections) {←Loopissa käydään yhdistetyt johdot
    foundConnections = foundConnections - 1;
    for(var i in wireArray[offset].connectedWires) {
      var _wire = wireArray[offset].connectedWires[i];
      if(!loopedArray.includes(_wire)) {
        loopedArray.push(_wire);
        wireArray.push(_wire);
        foundConnections++;
        if(_wire.on)
          state = true;←Ulkoinen ON tila eri johdolta
        }
      }
      offset++;
    }
    for(var i in wireArray) {
      wireArray[i].connectedon = state;←Ulkoinen ON tila
    }
  }
```

Esimerkkikoodi 9. Johtojen yhteneväisyyden ohjelmointi

6.2 Function block -palikoiden kopiointi

6.2.1 Ongelman selitys

Osa ohjelmoitavista ongelmista tulee vastaan ohjelmaa tehdessä. Näihin ennalta odottamattomiin ongelmiin usein kuitenkin löytyy helppo ratkaisu.

Jotta saataisiin helppokäyttötoiminto ohjelmoitavien palikoiden kopioimiseen, palikat täytyi muuttaa tallennettavaan muotoon. Helpoin ratkaisu tähän on käyttää JSON.Stringify-funktiota (2), joka muuttaa objektin tekstimuotoon, jonka voi lopulta kopioida leikepöydälle. Jokaiseen function blockiin on tallennettu, mihin johtoon tämä function block on kiinnitetty. Myös nämä johdot muistavat oman function blockinsa. JSON.Stringify-funktio ei pysty muuttamaan tällaista tilannetta, sillä se tekisi loputtoman pitkän tekstin objektien toisiinsa viittauksista.

6.2.2 Ratkaisu

Objektinotaatiot muutettiin ennen kopiointia funktiolla muotoon, jossa ei ollut muuta tietoa kuin objektin ID ja tyyppi. Malliratkaisu on Esimerkkikoodi 10:ssä.

```
var _copystring = JSON.stringify( _copyArray, function( key,
value) {
    if( key == 'inputs' || key == 'outputs' || key == 'connect-
edWires') {
        var _arr = [];
        for (key in value) {
            _arr.push({id:value[key].id,type:value[key].type})
        }
        return _arr;
    }else if(key =='input' || key == 'output' || key ==
'source') {
        var _arr = [];
        _arr.push({id:value.id,type:value.type})
        return _arr;
    }else {
        return value;
    }
},1)
```

Esimerkkikoodi 10. Johtojen yhteneväisyyden ohjelmointi

7 Tulevaisuuden suunnitelmat

Tämä verkkosivusto ei ole vielä mielestäni valmis. Visioni lopullisesta sivustosta on paljon laajempi kuin, mitä opinnäytetyönä olisi realistista valmistaa. Verkkosivuston valmistuminen edellyttää muutaman vuoden täysipäiväistä ohjelmointia. Nykyisessä mallissa on ainoastaan function block diagram -ohjelmointikielenä, mutta tavoitteenani on lisätä tälle sivustolle myös kaikki muut standardisoidut ohjelmointikielet.

Haluaisin myös lisätä tähän työtilan ja sivupalkin lisäksi kolmannen näkymän. Tässä näkymässä olisi simuloitu tehdasikkuna, jonka voisi myös muokata harjoituksille soveltuvasi. Tämä tehdasikkuna olisi myös hyvä simuloida oikein, eli logiikkaan voisi myös lisätä IO-mappauksen ja GVL-yhdistyksen kyseiselle mappaukselle. Tehdasikkunan tarkoituksena olisi esittää realistinen simulaatio tehtaasta tai tehtaan osasta.

Minulla on myös kehittämisideoita, joiden avulla harjoituksista saisi vielä mielenkiintoisempia. Yhtenä esimerkkinä olisi harjoitus, jossa opiskelijat kilpailisivat toisiaan vastaan. Tässä harjoituksessa tehdasnäkymänä olisi 2D-autorata, jossa opiskelijat automatisoisivat vaihteiden vaihdon, auton ohjauksen sekä kaasun ja jarrun käytön. Suoritukset listattaisiin, jolloin opiskelijat näkisivät, kuka heistä on nopein.

Tarkoituksena on myös parantaa opettajan työkaluja. Harjoitukset tulevat kurssimallisesti esiteltyinä verkkosivustolle, jossa opettajat voivat tarkastella opiskelijoiden suorituksia kyseisissä tehtävissä ja kohdentaa opetusta tarpeen mukaisesti. Ohjelmaan on mahdollista lisätä myös arviointityökalut.

Opinnäytetyön valmistuttua aion jatkaa tätä projektia ja saada tästä mahdollisesti kaupallisen version, mikäli oppilaitoksilla on tähän kiinnostusta.

Tämä oli ensimmäinen suuri ohjelmointiprojektini. Oli mielenkiintoista nähdä, miten sain kaikki pienet verkkosivuston komponentit toimimaan yhdessä. Tämä vaati ohjelman uudelleenkirjoittamista moneen otteeseen sekä kokonaisuutena että pienempinä osina. Projekti vei kokonaisuudessaan noin 200 tuntia ohjelmoidessa, mutta suunnitteluun, testaamiseen ja aiheen opetteluun kului huomattavasti enemmän aikaa.

Suurin osa koodista on optimoitu työn edetessä ja muokattu helppokäyttöisemmäksi muissa osissa koodia. Opin paljon ohjelmoinnin järjestämisestä ja jaottelusta, joten nykyiseen versioon on helppo lisätä suunnittelemani lisäykset.

Opin erityisesti function block -diagrammin ohjelmoinnista. Harjoitusten luonti on haastavaa ja niitä miettiessäni löysin uusia keinoja, joilla voin rakentaa erilaisia ominaisuuksia. Esimerkiksi oivalsin, että SR-piirin voi myös tehdä AND- ja OR-palikkaa käyttäen.

Olen tyytyväinen työn lopputulokseen. Opinnäytetyöni täyttää sille asettamani tavoitteet ja uskon, että tästä saa vielä hyvän työkalun opiskeluun.

Lähteet

1 Riihimaa, Jaakko, 2016. EU tietosuoja-asetus vaikuttaa AMKien innovaatiotoimintaan. Verkkoaineisto. Uasjournal. <https://uasjournal.fi/tag/digitalisaatio/>. 9.3.2016. Luettu 21.2.2018.

2 Scheinin, Patrik, 2015. Digitalisaatio mullistaa oppimisen. Verkkoaineisto. Helsinki. <https://www.helsinki.fi/fi/uutiset/opetus-ja-opiskelu/digitalisaatio-mullistaa-oppimisen>. 1.6.2015. Luettu 20.2.2018.

3 Härkönen, Erkki, 2017. Digitalisaatio opetuksen ja oppimisen apuna. Verkkoaineisto. Utu. <https://blogit.utu.fi/utu/2017/11/17/digitalisaatio-opetuksen-ja-oppimisen-apuna/>. 17.11.2017. Luettu 24.2.2018.

4 PHP-Manuaali. Verkkoaineisto. W3schools. <https://www.w3schools.com/php/>. Luettu 12.3.2018.

5 Javascript-manuaali. Verkkoaineisto. W3schools. <https://www.w3schools.com/js>. Luettu 10.12.2017.

6 HTML-manuaali. Verkkoaineisto. W3schools. <https://www.w3schools.com/html/>. Luettu 1.2.2018.

7 CSS-manuaali. Verkkoaineisto. W3schools. <https://www.w3schools.com/css/>. Luettu 12.12.2017.

8 Rouse, Margaret, 2008. Object-oriented programming (OOP). Verkkoaineisto. Techtarget. <https://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP>. 7.8.2008. Luettu 10.6.2017.

9 IEC 61131-3. basic software architecture and programming languages of the control program within PLC, subcommittee SC 65B of Technical Committee TC65. Luettu 20.11.2017.

Tilastotietoa projektista

Työtunnit:

Clientside 156 h

Serverside 62 h

Yhteensä 218 h (Ei sisällä tunteja aiheen opiskelusta tai projektin suunnittelusta)

Koodin pituus:

Suurin tiedosto: 684 riviä ja 27236 merkkiä, joka sisälsi FBD-työkalut.

Koodin kokonaispituus: 2525 riviä ja 82803 merkkiä.