Satakunnan ammattikorkeakoulu

Joona Vaurio

OPEROINTIPANEELI WWW-PALVELIMENA

Tekniikka ja merenkulku Pori

Tietotekniikan koulutusohjelma

Ohjelmointitekniikan suuntautumisvaihtoehto

2008

OPEROINTIPANEELI WWW-PALVELIMENA

Vaurio, Joona
Satakunnan ammattikorkeakoulu
Tekniikka ja merenkulku Pori
Tietotekniikan koulutusohjelma
Lokakuu 2008
Työn ohjaaja: Aarinen, Reino
UDK: 004.5, 004.738
Sivumäärä: 50

Asiasanat: operointipaneeli, web-käyttöliittymä, www-palvelin

---

Tämän opinnäytetyön aiheena oli tehdä web-käyttöliittymä ohjelmoitavassa logiikassa toimiviin muisti- ja nopeuspeleihin. Järjestelmän operointipaneelissa on sisäänrakennettu www-palvelin-toiminto, johon web-käyttöliittymät oli tarkoitus sijoittaa. Tarkoituksena oli selvittää ohjelmoitavan logiikan ja operointipaneelin www-palvelimen välistä yhteyttä, mutta työtä ei ollut mahdollista toteuttaa halutulla tavalla. Peleissä värejä oli määrä vaihdella ohjelmoitavan logiikan mukaan reaaliajassa web-sivuilla, mutta se ei ole mahdollista, koska HTTP-protokolla on tilaton. Tästä johtuen ohjelmoitava logiikka jouduttiin jättämään pois. Operointipaneelin www-palvelimen ominaisuudet olivat myös todella heikot, eikä pelien tuloksia voinut tallentaa palvelimelle, niin kuin alun perin oli tarkoitus. Toteutin lopulta pelit operointipaneelin www-palvelimeen JavaScriptillä ja parhaiden tulosten tallennus toteutettiin erilliselle PHP-palvelimelle.

OPERATION PANEL AS A WEB SERVER

Vaurio, Joona
Satakunta University of Applied Sciences
Facility of Technology and Maritime Management Pori
Degree Programme in Information Technology
October 2008
Supervisor: Aarinen, Reino
UDC: 004.5, 004.738
Number of Pages: 50

Key Words: operation panel, web based user interface, web server

---

The purpose of this thesis was to create a web based user interface for memory and speed games that function in a programmable logic unit. The operation panel included in the system has a built-in web server where the web pages can be uploaded. The purpose was to find out about the communication between the web server of the operation panel and the programmable logic unit. However, it was not possible to implement the user interface for the games this way. The games were supposed to change colors in the web site in real time and it can not be done because HTTP is a stateless protocol. Thus, the programmable logic had to be left out. The web server of the operation panel has extremely limited features and it was not even possible to store the highest scores from the games to the web server, which was originally the plan. In the end I programmed the games using JavaScript and the highest scores were saved to an external PHP-server.

LYHENTEET

AJAX - Asynchronous JavaScript And XML

CGI – Common Gateway Interface

CSS – Cascading Style Sheets

DHTML – Dynamic HTML

FTP – File Transfer Protocol

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

IP – Internet Protocol

JS – JavaScript

OSI – Open Systems Interconnection

PC – Personal Computer

PHP – PHP:Hyperext Preprocessor

SSI – Server Side Include

TCP – Transmission Control Protocol

WWW – World Wide Web

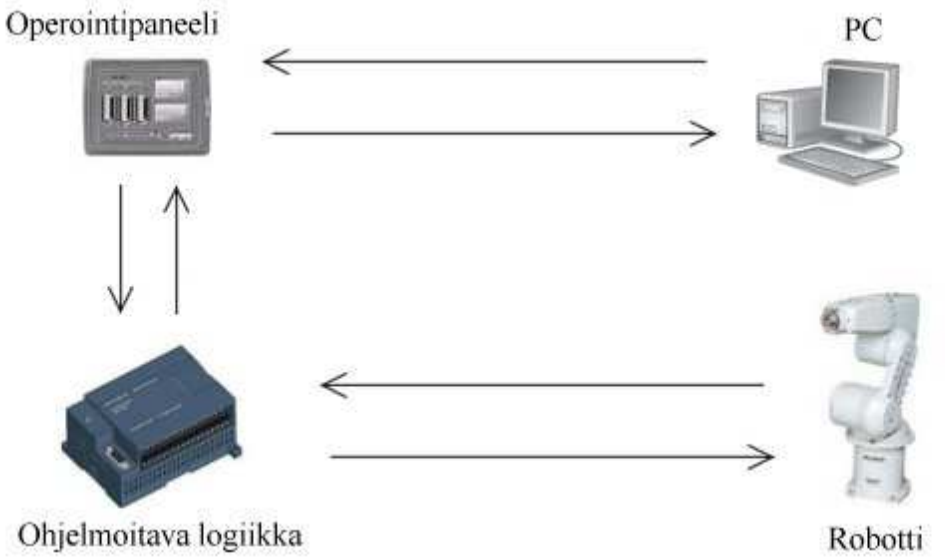
XHTML – eXtensible HyperText Markup Language

XML – eXtensible Markup Language

SISÄLLYSLUETTELO

1 JOHDANTO

Työ liittyy Satakunnan ammattikorkeakoulun Tekniikan ja merenkulun toimialan tekniikkanäyttelyn pääkohteeksi tulevaan robottiohjattuun biljardipeliin. Biljardipallojen pudotessa pussiin ne tunnistetaan älykameralla, jonka jälkeen ohjelmoitava logiikan avulla robotti osaa siirtää ne oikeaan paikkaan pallotarjottimella. Robotin yhteyteen haluttiin myös muisti- ja nopeuspeli, jotka sijaitsevat ohjelmoitavassa logiikassa. Opinnäytetyössä tarkoitus oli tehdä muisti- ja nopeuspelille web-käyttöliittymät, jolloin pelejä voisi pelata verkon kautta. Web-sivut sijoitetaan järjestelmän operointipaneeliin, joka voi toimia www-palvelimena. Parhaat tulokset voi tallentaa ja ne ovat kaikkien nähtävillä web-käyttöliittymässä. Ohjelmoitavaan logiikkaan sijoitettavien pelien ohjelmointi ei sisältynyt työhön, mutta pelien toiminta logiikan ja käyttöliittymän välillä oli kuvattava niin, että pelien toteutus myöhemmin olisi mahdollista.



Kuva 1. Järjestelmässä tietokone (selain) pyytää sivua operointipaneelin www-palvelimelta, joka hakee pelitiedot ohjelmoitavalta logiikalta ja lähettää web-sivun tietokoneelle (selaimelle) vastauksena.

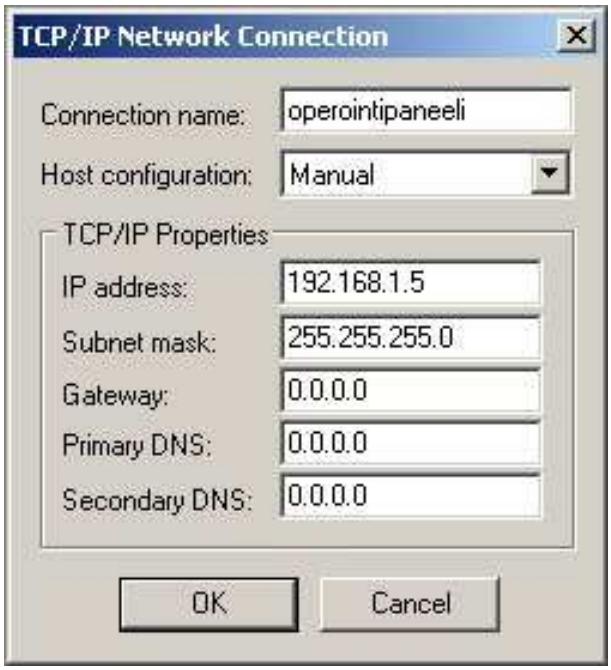
2 ESISELVITYS

Esiselvityksessä tehtävänä oli selvittää operointipaneelin www-palvelimen ominaisuuksia ja tutkia, onko työn toteuttaminen halutulla tavalla mahdollista. Ensimmäinen työn määrittelyyn liittyvä ongelma oli se, että muisti- ja nopeuspeleissä värien pitäisi vaihtua web-sivulla ohjelmoitavan logiikan määräämässä tahdissa. HTTP-protokolla, jonka avulla web-sivujen siirto verkossa tapahtuu, on tilaton, eikä web-sivun sisältöä voida oletusarvoisesti reaaliajassa muuttaa palvelimen toimesta.

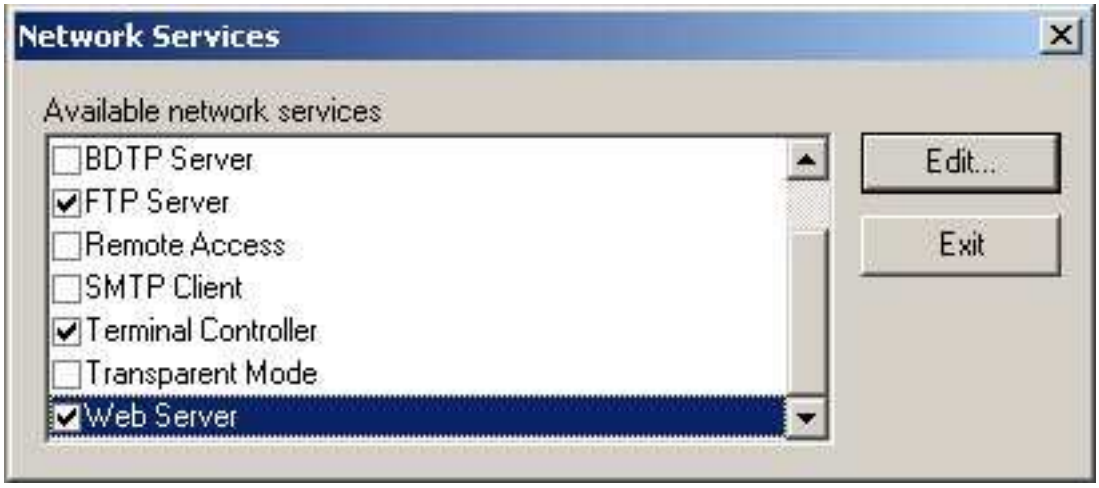
2.1 Operointipaneeli

Aluksi oli tutustuttava Beijer Electronicsin toimittamaan operointipaneeliin. Operointipaneeli on E1000-tuotesarjaan kuuluva kosketusnäytöllinen E1101-malli. Laite toimii 24V tasajännitteellä, joten siihen tarvittiin muuntaja, joka löytyi koululta. Käynnistyksen jälkeen operointipaneelissa oli näkyvillä ainoastaan minimaalinen valikko, josta sai vaihdettua IP-osoitteen. Työn aikana operointipaneelin IP-osoitteena käytettiin osoitetta 192.168.1.5. IP-osoitteen asettamisen jälkeen yritys saada operointipaneelin tietokoneen välille yhteys operointipaneelissa olevan ethernet-portin kautta ei kuitenkaan onnistunut. Jonkin ajan kuluttua kävi selväksi, että operointipaneelista ei voi käyttää pelkkää www-palvelinta, vaan siihen on ladattava ohjelmaprojekti. Tämä johtuu siitä, että www-palvelin on asetettava päälle projektin ominaisuuksista. Projektit luodaan operointipaneeliin E-Designer-nimisellä PC-ohjelmalla, joka on saatavilla Beijer Electronicsin web-sivuilta latausavaimen avulla.

Ohjelmalla voi luoda monimutkaisia ohjelmia automaatiojärjestelmien hallintaan. Työssä tarvittiin kuitenkin vain www-palvelinta, joten vain sen tarvitsemat asetukset oli asetettava. Projektin verkkoasetuksiin (Network Connections, Kuva 2) piti määrittää operointipaneelin IP-osoite, jotta projektin voi siirtää operointipaneeliin. Tämän lisäksi www-palvelin piti asettaa päälle verkkopalvelut-valikosta (Network Services, Kuva 3).

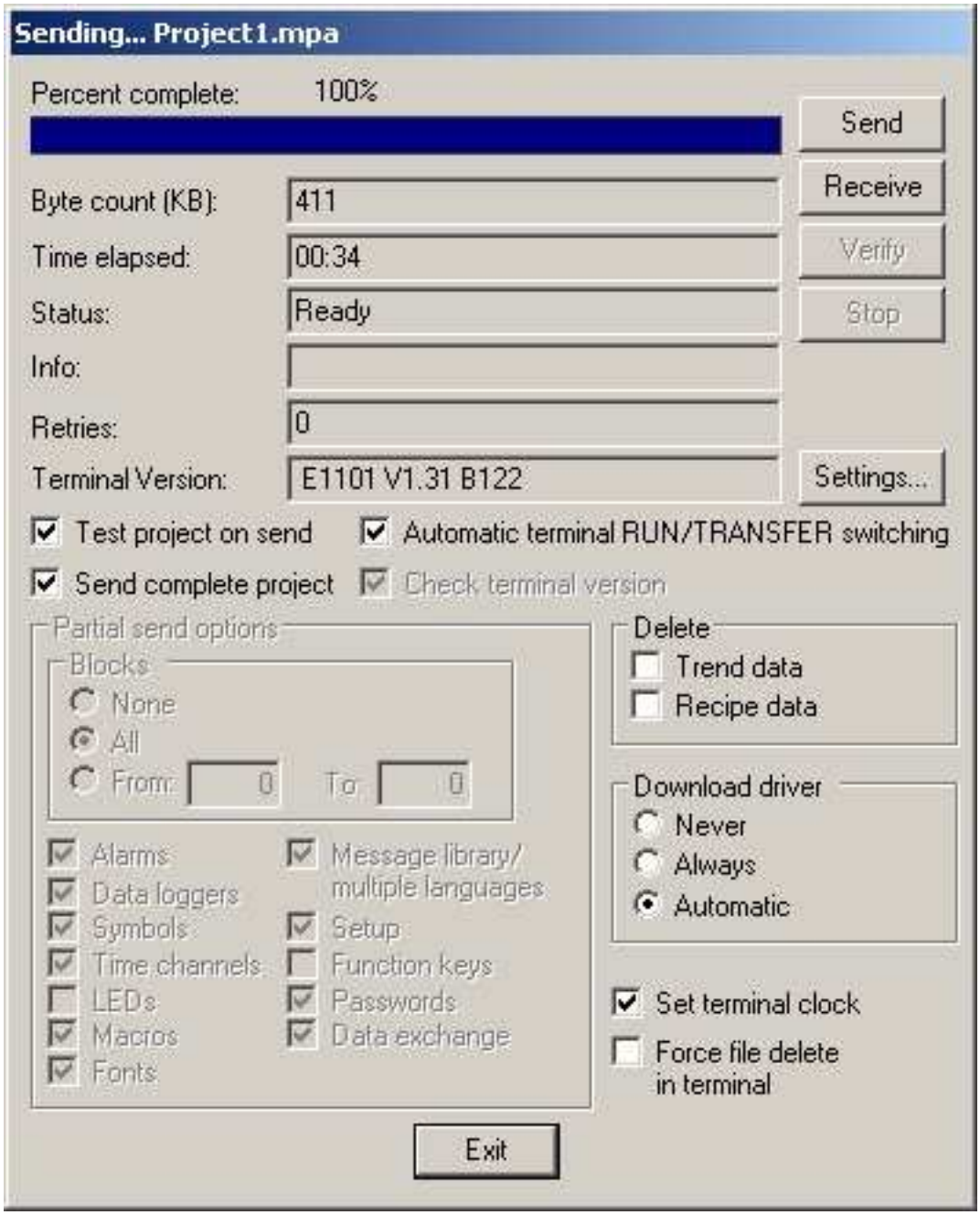


Kuva 2. E-Designer-projektin valikko verkkoasetuksille.



Kuva 3. WWW-palvelimen päällekytkentä tapahtui verkkopalvelut-valikosta. Edit-kohdassa oli vain portin vaihtomahdollisuus ja mahdollisuus suojata web-sivut salasanalla. Muita asetuksia operointipaneelin www-palvelimella ei ole.

Verkkopalvelut-valikosta onnistuu myös FTP-palvelimen päällekytkentä. FTP-palvelimen avulla voi helposti siirtää tiedostoja, kuten web-sivuja, operointipaneelin ja tietokoneen välillä. Tämän jälkeen projektin siirto operointipaneeliin onnistui ethernet-liitännän kautta siirtovalikosta (Kuva 4), jonka sai auki valitsemalla *Transfer > Project...*



Kuva 4. Projektin siirtovalikko E-Designer-ohjelmassa. Projektin voi siirtää ohjelmasta operointipaneeliin tai operointipaneelista ohjelmaan (Send / Receive).

Onnistuneen siirron jälkeen operointipaneelin ruudulle ilmestyi E-Designerillä tehty projekti näkyviin. Koska oma projektini oli pelkkä tyhjä projekti, josta vain muutamaa asetusta oli muutettu, ruudulla näkyi ainoastaan harmaa tausta. IP-osoitetta ei enää voinut muuttaa ruudulta, sillä valikkoa ei enää saa näkyviin sen jälkeen, kun operointipaneeliin on siirretty projekti. IP-osoitteen vaihto onnistuu tällöin muuttamalla operointipaneelin muistikorttiluukun sisällä olevia tilakytkimiä. Tilakytkimiä on neljä kappaletta ja normaalitilassa ne ovat kaikki pois päältä. Kytkemällä ensimmäisen kytkimen päälle, pääsee vaihtamaan IP-osoitetta. Tilakytkimien avulla esimerkiksi operointipaneelin tehdasasetuksien palautus on mahdollista. Operointipaneeli on sammutettava kytkimien vaihdon ajaksi. Projektin siirron jälkeen www-palvelimen ollessa päällä selaimella pääsi operointipaneelin web-sivuille sen IP-osoitetta käyttäen. Selaimeen aukesi paneelissa valmiina oleva oletussivu, joka sisälsi yleistä tietoa paneelista (Kuva 5).

Kuva 5. Operointipaneelin oletussivu sisälsi yleistietoa paneelista, kuten käynnistyskerrat, käyntiajan ja vapaan muistin määrän.

FTP (File Transfer Protocol) on tiedostonsiirtoprotokolla, jonka avulla tiedostoja voi siirtää kahden tietokoneen välillä. FTP on käyttöjärjestelmäriippumaton. FTP-yhteyden muodostus operointipaneeliin onnistui käyttäen Internet Explorerin sisäänrakennettua FTP-ohjelmaa, jonka jälkeen operointipaneelin tiedostoihin pääsi käsiksi (Kuva 6).

Kuva 6. Operointipaneelin hakemistot.

HTML-hakemisto on työn kannalta ainoa tärkeä hakemisto. Sinne sijoitetaan kaikki www-palvelimen web-sivut. Oletuksena hakemisto sisälsi ainoastaan INDEX.HTML-tiedoston. Se näytetään käyttäjälle, mikäli mitään erityistä sivua ei ole pyydetty.

2.1.1 WWW-palvelin

Yksinkertaisimmillaan www-palvelin lähettää vain pyydetyn staattisen resurssin, kuten HTML-sivun tai kuvan, selaimelle sellaisenaan, eikä käsittele resurssia millään tavalla. Kehittyneemmät palvelimet ovat niin sanottuja sovelluspalvelimia, jotka suorittavat ensin pyydetyn ohjelman ja lähettävät sen tuottaman tuloksen selaimelle. Tällaisia ohjelmia kutsutaan CGI-ohjelmiksi (Common Gateway Interface). CGI-ohjelmia käytetään paljon HTML-lomakkeiden tietojen lukuun ja käsittelyyn. Operointipaneelin www-palvelimessa on E-Designer-oppaan (Lähde 2) mukaan CGI- ja SSI-tuki. SSI (Server Side Include) on muuttuja, jonka arvon palvelin voi sisällyttää web-sivulle ennen sen lähetystä selaimelle.

INDEX.HTML-tiedoston siirto omalle tietokoneelleni onnistui FTP:tä käyttäen. Tiedostoa tutkimalla selvisi, että sivulla näkyvä operointipaneelin IP-osoite ja muut tiedot tulostuvat sivulle valmiilla CGI-ohjelmilla (Kuva 7).

```
17  IP address: <!--#exec cgi="/get_ipaddr.fn"--><BR>
18  <BR>
19  <BR>
20  <!--#exec cgi="/get_diag.fn"--><BR>
```

Kuva 7. Komento <!--#exec=”/get_diag.fn”--> tulosti sivulle suuren määrän tietoa operointipaneelista.

Minulle toimitetussa E-Designer-oppaassa oli listattuna joitakin SSI-muuttujia ja CGI-ohjelmia. Näillä onnistui operointipaneelin perusominaisuuksien, kuten päivämäärän ja signaalien arvojen tulostus ja asettaminen (Kuva 8). Scriptit kirjoitetaan HTML-kommenttien sisään, jolloin scriptin epäonnistuessa sivulle ei tulosteta mitään.

```
39  <form action="http://<!--#exec cgi="get_ipaddr.fn"-->/set_date.fn" method="post">
40  <input name="DD:MM:YYYY" />
41  <input type="submit" />
42  </form>
```

Kuva 8. Esimerkki paneelin valmiista CGI-scriptistä. Scriptiä on käytetty HTML-lomakkeen yhteydessä ja se vaihtaa paneelin päivämäärän. Päivämäärä on syötettävä muodossa DD:MM:YYYY.

Valmiit scriptit eivät kuitenkaan olleet opinnäytetyötä ajatellen juuri millään tavalla hyödyllisiä. Työssä oli tarve saada tehtyä omia CGI-scriptejä. Tästä asiasta ei kuitenkaan ollut minkäänlaista mainintaa yhdessäkään operointipaneelin esitteessä tai ohjeessa. Operointipaneelin www-palvelimesta ei löytynyt juuri mitään tietoa verkosta. Operointipaneelista yleensäkin löytyi todella vähän tietoa muualta, kuin Beijer Electronicsin omilta sivuilta, mutta sielläkään ollut www-palvelimeen liittyen minkäänlaista tarkempaa tietoa.

SSI-kielessä on joitakin perusmuuttujia, kuten tiedoston muokkausajankohta ja käyttäjän selain. Ne eivät kuitenkaan toimineet lainkaan, joka oli todella yllättävää koska tuki SSI:lle oli mainittu. Myöskään SSI-kielellä normaalisti mahdollinen toisen sivun sisällyttäminen sivuun ei toiminut. Tämän vuoksi navigointivalikkokin on tehtävä jokaiseen sivuun erikseen, eikä sitä voi sisällyttää samasta tiedostosta jokaiseen sivuun.

Kuva 9. Vasemmalla on selaimessa näkyvä sivu, keskellä on selaimesta otettu sivun lähdekoodi ja oikealla on todellinen lähdetiedosto. Valmiit scriptit toimivat hyvin, mutta normaalit SSI-muuttujat eivät tulosteet sivulle.

Epäselvää oli pitkän aikaa se, voiko paneeliin tehdä omia CGI-ohjelmia. Asia kuitenkin selvisi tiedusteltuani asiaa Beijer Electronicsin asiantuntijoilta. He eivät osanneet vastata kysymykseen ja tiedustelivat asiaa puolestani jopa Ruotsista asti, jossa asiasta oli tarkoitus tietää enemmän. Saamassani vastauksessa oli kuitenkin liitteenä vain sama E1000-sarjan E-Designer-opas, joka minulla jo oli ja HTML-esimerkkitiedostoja, joissa käytettiin ainoastaan operointipaneelin valmiita fn-päätteisiä scriptejä. Tämän jälkeen oli todettava, että mahdollisuutta omien scriptien tekoon ei nähtävästi ole. Hyvästä ensivaikutelmasta huolimatta operointipaneelin www-palvelin osoittautui hyvin alkeelliseksi, eivätkä sen ominaisuudet täyttäneet odotuksia. Toisaalta tällaisessa operointipaneelissa ei normaalisti ole tarvettakaan ylläpitää monimutkaisia sivustoja, joten se on ymmärrettävää. Operointipaneelin www-palvelin on luultavasti tarkoitettu pääosin järjestelmien seurantaa varten. Operointipaneelin tekniset tiedot ovat liitteenä (Liite 2 ja Liite 3).

2.2 HyperText Transfer Protocol (HTTP)

HTTP on tiedonsiirtoprotokolla, jolla tieto verkossa siirretään. HTTP-tiedonsiirrossa asiakasohjelma avaa TCP-yhteyden ja lähettää pyynnön (REQUEST) resurssista palvelimelle, joka lähettää asiakasohjelmalle takaisin vastauksen (RESPONSE). Yleisin protokollan käyttö on web-sivujen siirto verkossa, jolloin asiakasohjelmana toimii selain ja palvelimena jokin www-palvelin. Kun käyttäjä kirjoittaa selaimen osoitekenttään jonkin www-osoitteen, selain lähettää pyynnön halutusta sivusta kyseiseen osoitteeseen. Palvelin vastaa lähettämällä selaimelle pyydetyn sivun, joka sisältää yleensä HTML-koodia. Yleisesti internet-sivuja selailtaessa puhutaan, että mennään jollekin sivulle. Teoriassa sivulle ei kuitenkaan mennä, vaan sivu haetaan palvelimelta paikalliselle koneelle.

Kuva 10. HTTP toimii OSI-mallin ylimmässä kerroksessa.

Pyynnön ja vastauksen rakenne on samanlainen:

```
Alustusrivi / Statusrivi
Otsikkorivit
[Pakollinen tyhjä rivi]
Viestirunko
```

Pyynnön alustusrivi:

`GET resurssi protokolla/versio, esim.` **`GET /sivu.html HTTP/1.1`**

Vastauksen statusrivi:

`protokolla/versio statuskoodi viesti, esim.` **`HTTP/1.1 404 Not Found`**

Statuskoodeja on useita, mutta ne voidaan luokitella ensimmäisen numeron perusteella seuraavanlaisesti:

```
1xx   Informaatiota           esim. 100=Continue
2xx   OK                      esim. 200=OK
3xx   Uudelleenohjaus         esim. 301=Moved Permanently
4xx   Virhe selainpäässä      esim. 401=Unauthorized
5xx   Virhe Palvelinpäässä    esim. 500=Internal Server Error
```

Kun palvelin on lähettänyt asiakasohjelmalle pyydetyn resurssin, se sulkee yhteyden, eli HTTP on tilaton protokolla. Toisin sanoen, kun sivu on latautunut selaimeen, se on staattinen. Sivua ei voida päivittää tai eri sivulle ei voida siirtyä ennen, kuin uusi pyyntö lähetetään palvelimelle. Sivun latautumisen jälkeen asiakasohjelmalla ja palvelimella ei ole mitään tietoa toisistaan, eli seuraavan pyynnön tulos ei riipu edellisestä pyynnöstä. Poikkeuksena tähän on kehitetty istunnot (SESSION), joiden avulla palvelin voi tunnistaa käyttäjän jokaisen pyynnön yhteydessä. Istuntojen avulla on täten mahdollista toteuttaa esimerkiksi sisäänkirjautuminen sivustolle.

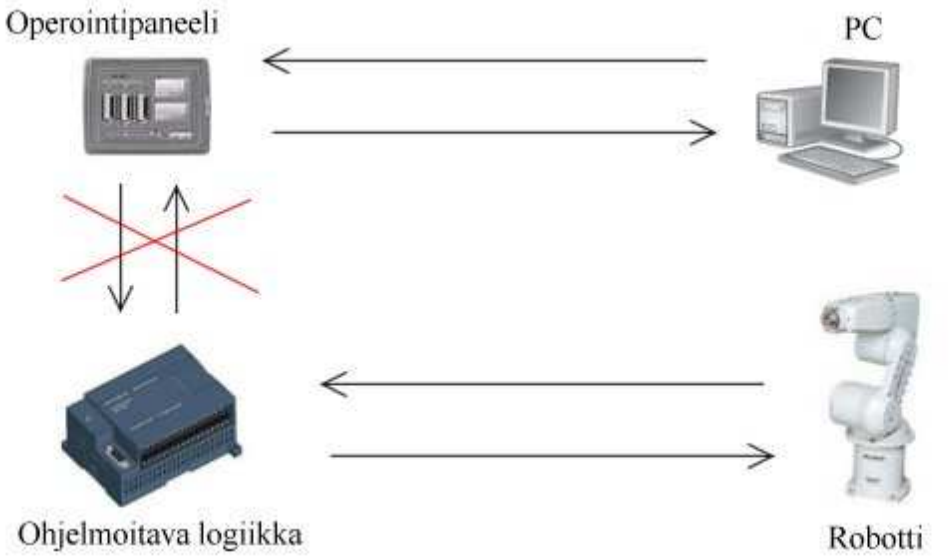
2.3 Työn uudelleenmäärittely

Työn toteutukseen liittyen oli monta ongelmaa. Koska HTTP-protokolla on tilaton, eikä jatkuvaa yhteyttä palvelimeen ole olemassa, ohjelmoitava logiikka ei voi reaaliajassa vaihdella pelissä näytettäviä värejä web-sivulla. Mahdollisuutta käyttää ohjelmoitavaa logiikkaa JavaScriptin kanssa pohdittiin myös jonkin verran. Ajax on melko uusi tekniikka, jonka avulla selain osaa JavaScriptillä hakea, ikään kuin taustalla, tietoa palvelimelta päivittämättä koko sivua uudelleen. Ajax ei vaadi www-palvelimelta mitään, mutta selaimen on tuettava sitä. Ajaxia käytetään paljon esimerkiksi tekstikenttien automaattiseen kirjoituksentäyttöön. Yksinkertaisella ohjelmalla operointipaneelin www-palvelimella sijaitsevan tekstitiedoston sisällön sai haettua HTML-sivuun lataamatta koko sivua uudelleen. Tällä tekniikalla olisi voinut olla mahdollista hakea ohjelmoitavalta logiikalta aina seuraavan signaalin arvo eli pelin väri. Tähänkin liittyy silti ongelmia, koska signaalin arvon hakeminen ohjelmoitavalta logiikalta voi kestää hyvin kauan. Varsinkaan nopeuspelissä minkäänlainen viive ei ole hyväksyttävää.

Flashin käyttämisestä oli myös alustavia ajatuksia. Flash on HTML-sivulla oleva upotettu objekti, joka vaatii toimiakseen vain sen, että selaimeen on asennettu Flash-tuki. Toteutus olisi mennyt niin, että Flash-pelit olisivat vaihtaneet tietoa suoraan ohjelmoitavan logiikan kanssa ilman www-palvelinta välissä. Tämän toteutusmahdollisuuksista ei ollut kuitenkaan tietoa, eikä sitä mietitty sen enempää.

Yhtenä suurena ongelmana oli myös se, että ohjelmoitava logiikka voi pitää vain yhtä peliä päällä kerrallaan. Toisin sanottuna vain yksi henkilö kerrallaan olisi voinut pelata. Pelien toteutus ohjelmoitavan logiikan kanssa olisi näin ollen todella epäkäytännöllistä.

Koska operointipaneeliin ei voinut omia scriptejä tehdä, pelejä ei sinnekään voinut ohjelmoida ja siinäkin tapauksessa värien vaihto olisi ollut tehtävä Ajaxin avulla. Näin ollen päädyimme asian suhteen sellaiseen ratkaisuun, että pelien ohjelmointi sisältyy opinnäytetyöhön ja ne ohjelmoidaan JavaScriptillä. Näin ollen pelit toimivat täysin paikallisesti selaimessa ja ohjelmoitava logiikka jätetään työstä kokonaan pois. Tämä jäi ainoaksi käytännölliseksi toteutustavaksi. Tällöin kuitenkin alkuperäinen tarkoitus käyttää ohjelmoitavaa logiikkaa www-palvelimen kanssa jäi pois ja toteutuksesta tuli paljon yksinkertaisempi.



Kuva 11. Ohjelmoitava logiikka jätettiin pelien toteutuksesta pois.

3 OHJELMOINTIKIELET

Koska operointipaneeliin ei voinut ohjelmoida omia CGI-scriptejä, käytettävät ohjelmointikielet jäivät vähäiseksi. Käytössä olivat vain web-sivujen peruskielet HTML, CSS ja JavaScript. Nämä kaikki ovat palvelinriippumattomia ja vaativat enemmänkin selaimelta ominaisuuksia kuin www-palvelimelta.

3.1 HyperText Markup Language (HTML)

HyperText Markup Language eli HTML on merkkauskieli, jolla kuvataan web-sivujen rakenne. Ensimmäinen versio HTML:stä ilmestyi vuonna 1990. Alun perin rakenteen kuvaus oli tarkoitettu HTML:n ainoaksi tarkoitukseksi. Kuitenkin, kun 1990-luvun alkupuolella sivuja tehtiin käyttäen pelkästään HTML-kieltä, tarve muokata sivujen ulkoasua sai selainvalmistajat lisäämään uusia ominaisuuksia HTML:ään. Lisäykset eivät tietenkään kuuluneet viralliseen HTML:n määrittelyyn, mutta osa niistä päätyi siihen ajan myötä. Näin kieli laajeni käsittämään myös sivujen visuaalista puolta. Koska selainvalmistajat tekivät omia määritelmiään, sivut saattoivat näyttää hyvinkin erilaisilta eri selaimilla katseltuna. HTML käsittää käytännössä vain joukon ennalta määrättyjä elementtejä, joita kutsutaan myös tageiksi.

Kielen standardointi on kuitenkin ajan myötä poistanut vanhoja tageja ja lisännyt joitakin uusia. HTML:n standardoinnista vastaa World Wide Web Consortium (W3C). Kehityksen myötä HTML-dokumentin alussa on dokumentin määrittely (Document Type Definition, DTD). Määrittelyjä on kolme tasoa: strict, transitional ja frameset. Strict on tiukka standardin mukainen määrittely, josta on poistettu vanhentuneet tagit. Strictin mukaan tehdyn sivun toimivuus eri selaimissa on täten parempi. Käytännön toteutus strictin mukaisesti aiheuttaa kuitenkin yleensä päänvaivaa ja tämän vuoksi transitional, joka ei ole niin tarkka tagien käytöstä, on paljon käytetympi. Frameset-määrittelyä ei juurikaan enää käytetä, vaikka se sallii sivun jakamisen useampaan osaan.

HTML:stä tehtiin XML-syntaksin toteuttava versio eXtensible HyperText Markup Language (XHTML), joka eroaa HTML:stä muun muassa siten, että jokaisen tagin

nimi on oltava pienellä kirjoitettu ja jokainen tagi on suljettava. Esimerkiksi HTML:n mukaan rivinvaihdon suorittavalla tagilla ei ole lainkaan sulkevaa tagia, mutta XHTML:n mukaan se on oltava. Näin ollen tagista tehdään niin sanotusti itsesulkeva. Rivinvaihto on HTML:ssä
 ja XHTML:ssä
.

Lähes kaikki HTML-koodi, tehdään nykyisin XHTML:n mukaisesti, niin kuin tässäkin työssä. XHTML -dokumentin yleisrakenne koostuu dokumentin määrittelystä ja vaadituista perustageista html, head ja body. HTML-koodia voi kirjoittaa millä tahansa tekstieditorilla ja kaikki selaimet osaavat tulkita sitä.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4  <html xmlns="http://www.w3.org/1999/xhtml">
5  <head>
6   <title>Sivun otsikko</title>
7  </head>
8
9  <body>
10  <!-- Tämä on HTML-kommentti -->
11  <h1>Hei maailma!</h1>
12 </body>
13 </html>
```

Kuva 12. Esimerkki yksinkertaisesta XHTML -sivusta transitional -määrittelyä käyttäen.

Tämä yksinkertainen koodi tulostaisi sivulle tekstin ”Hei maailma!”. Sivun varsinainen sisältö tulee body-tagin sisälle. HTML-sivujen tiedostopääte on html. Oletussivu, jolle käyttäjä ohjataan aina ensimmäiseksi on yleensä nimeltään index.html. HTML-kielellä voi tehdä kuitenkin ainoastaan staattisia sivuja, eikä nykyään pelkästään sitä käyttäen enää tehdä sivustoja. Tarve dynaamiseen sisältöön sekä muihin toiminnallisuuksiin, kuten matemaattisiin toimenpiteisiin, on tuottanut useita ohjelmointikieliä, joilla nämä onnistuvat.

Nämä ohjelmointikielet, kuten PHP, tarvitsevat palvelimen käsittelemään sivut ennen, kuin ne näytetään käyttäjille. Sivulla voi olla monimutkaisiakin ehtolauseita,

joiden perusteella sisältö tuotetaan, mutta lopputulos on aina pelkkää HTML:ää. Tämän voi todeta sivujen lähdekoodeja selaimella katsoen. HTML on niin perusasia, että siitä ei yleensä edes mainita. Esimerkiksi on hyvin yleistä kuulla, että jokin sivu on tehty PHP:llä. Todellisuudessa PHP -ohjelma tietenkin tuottaa lopputuloksena HTML-koodia. Olisi todella työlästä tuottaa suuria määriä tietoa sisältäviä taulukoita käsin kirjoittaen. Tämä voidaankin tehdä helposti jonkin ohjelmointikielen avulla iteroiden tietueet läpi. Näin ollen voidaan todeta, että kaikki sovelluspalvelimet ovat itse asiassa vain apuvälineitä HTML:n tuottamiseen.

3.2 Cascading Style Sheets (CSS)

HTML–sivujen ulkoasun muokkauksen tarve synnytti aikoinaan uusia HTML–tageja tähän tarkoitukseen, mutta HTML:stä alkoi tulla liian monimutkaista ja yhteensopivuus muodostui ongelmaksi. Tähän ongelmaan kehitettiin ratkaisuna CSS-kieli. CSS-tyyleillä sivun ja sen elementtien värit, fontit, kehykset jne. voidaan määritellä HTML-koodista erillisenä. CSS:n johdosta HTML on saavuttanut alkuperäisen asemansa pelkkänä sivun rakenteen määrityksenä. Tämä on merkittävä etu monimutkaisissa ja suurissa nykysovelluksissa, joissa sivun osa-alueet, kuten rakenne, tyylit ja sisältö, ovat eroteltuina sujuvampaa hallintaa varten.

CSS mahdollistaa tyylin määrittelyn yhteisenä kaikille elementeille, yhteisenä kaikille samantyyppisille elementeille ja yksittäiselle elementille. Selaimet tulkitsevat CSS-tyylit ja tämän vuoksi pieniä eroja eri selainten välillä tulee helposti. Jotkut tyylimääritykset eivät välttämättä toimi kaikissa selaimissa, joka on johtanut siihen, että joissakin tapauksissa sivustojen kehittäjät tekevät erilliset, selainkohtaiset tyylimäärittelyt sivuille.

CSS-tyyli voidaan määritellä kolmessa eri paikassa. Se voi olla HTML-elementin style-attribuuttina, head-tagin sisällä style-tagissa tai erillisessä CSS-tiedostossa. Erillinen tiedosto määritellään head-tagien sisällä.

3.3 JavaScript

JavaScript on ohjelmointikieli, jolla HTML-sivuun saadaan dynaamisuutta ilman sovelluspalvelinta. Sivulla olevaa HTML-koodia voidaan muuttaa lataamatta sivua uudelleen JavaScriptin avulla. On kuitenkin tärkeää ymmärtää, että JavaScript toimii asiakaskoneessa paikallisesti selaimen tulkitsemana. Tämä tarkoittaa sitä, että www-palvelin ei vaikuta mitenkään siihen, miten sivu muuttuu JavaScriptin aiheuttamana. Poikkeuksena on Ajax-tekniikka, jonka avulla selain osaa JavaScriptillä hakea palvelimelta tietoa ja päivittää vain halutun osan sivusta, samoin kuin Java Applet ja Flash voivat tehdä.

JavaScript-koodia voi sijoittaa mihin kohtaan tahansa HTML-tiedostossa <script> ja </script> tagien väliin. JavaScript-koodin voi myös laittaa erilliseen tiedostoon, jonka tiedostopääte on js. JavaScript tunnistaa HTML-sivulla tapahtuvia tapahtumia. Esimerkiksi, kun hiiri viedään tietyn elementin ylle tai kun jotakin klikataan, voi JavaScript suorittaa haluttuja toimenpiteitä. Tapahtumamäärittelyt tehdään suoraan kyseiseen HTML-tagiin ylimääräisellä attribuutilla. Normaaleita käyttötarkoituksia ovat mm. lomakkeiden tietojen tarkistukset ennen niiden lähettämistä palvelimelle, värien muuttaminen hiiren mennessä elementin ylle ja popup-ikkunoiden tekeminen. JavaScriptin ongelma on kuitenkin selain. Eri selainversiot voivat tulkita scriptiä eri tavoin ja joissakin selaimissa koodi ei välttämättä toimi ollenkaan.

```
<html>
<head>
<script language="JavaScript" src="scripti.js">
//ulkoisen tiedoston lisäksi tulevat koodit
function ilmoitus() {
    alert("Nappia on painettu");
}
</script>
</head>
<body>
<form><input type="button" onclick="ilmoitus()" /></form>
</body>
</html>
```

Kuva 13. Koodi tekee HTML-sivulle napin, jota painettaessa ruudulle tulee ilmoitus.

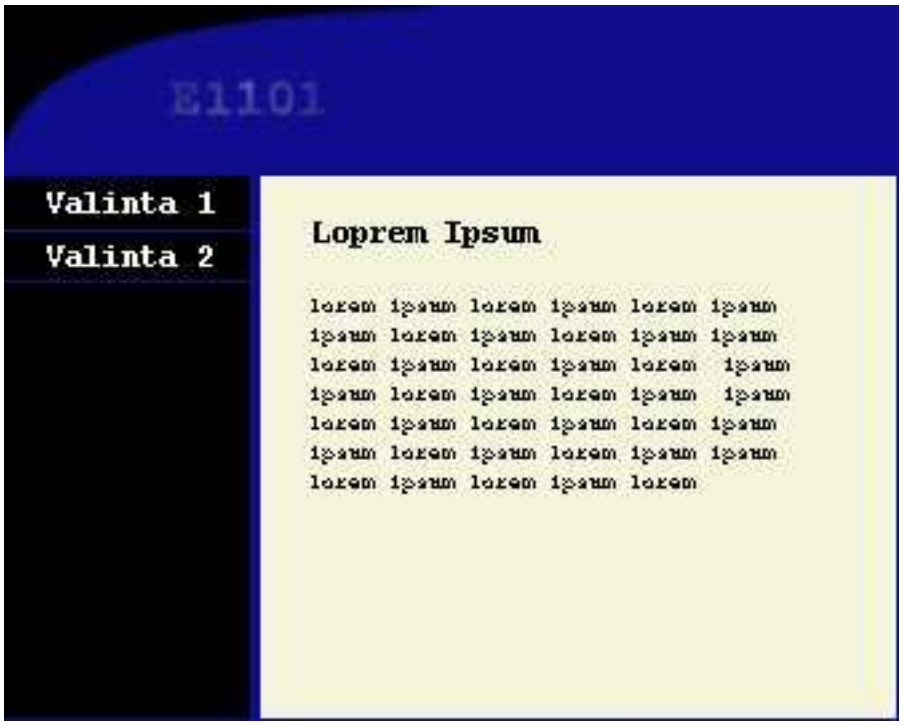
4 SUUNNITTELU

4.1 Käyttöliittymä

Käyttöliittymä on ohjelman osa, jonka kautta käyttäjä käyttää ohjelmaa. Monissa ohjelmistoissa käyttöliittymät ovat vielä nykyäänkin erittäin puutteellisia. Ohjelmoijat eivät juuri panosta helppokäyttöisyyteen. Heidän mielestään ohjelma on tietenkin helppo käyttää, koska he ovat sen itse ohjelmoineet ja tuntevat kaikki ohjelman ominaisuudet. He eivät välttämättä edes ymmärrä, että joku voikin yrittää suorittaa haluttua toimintoa eri tavalla: Ihmiset ajattelevat asioita eri tavoin. Jokaisella on varmasti kokemuksia siitä, miten yksinkertaisenkin perustoiminnon suorittaminen jossakin ohjelmassa on joskus melko hankalaa. Esimerkiksi Microsoftia on tästä asiasta moitittu jo pieni ikuisuus.

Käyttöliittymän on oltava passiivinen ja helppokäyttöinen. Värit eivät saa olla hyökkääviä tai toisiinsa sekoittuvia eikä sivuilla saa olla silmää rasittavia vilkkuvia komponentteja. Usein ihmisten tehdessä ensimmäisiä sivujaan ne ovat nimenomaan täynnä turhia vilkkuvia objekteja sekä raskaita värejä. Varsinkin web-sivustoilla suositellaan käyttämään vaaleita värejä, koska ne eivät ärsytä silmää. Web-käyttöliittymissä on myös muodostunut hyväksi havaittu perusmalli, jossa sivu koostuu bannerista, valikosta, pääosasta ja footerista. Tätä mallia noudattaa lähes jokainen sivusto ja ihmiset ovatkin tottuneet siihen, joten sitä kannattaa käyttää. Näin sivustojen käyttö sujuu rutiininomaisesti, vaikka juuri kyseisellä sivustolla ei olisi ennen käynytkään.

Tämän työn sivujen ulkoasun suunnittelussa tummansininen taustaväri nousi esille. Vaikka tummia värejä ei mielellään tulisi käyttää, se kuitenkin sopi työhön, koska pääosan taustaväriksi tulisi tasapainoksi jokin vaalea väri kuten beige. Banneriksi vasempaan yläreunaan olisi tarkoitus tulla musta kuvio tai palkki ja sen yhteyteen operointipaneelin mallimerkintä E1101 ikään kuin logona. Luonnoksen tekeminen sujui Paint Shop Pro 5 ohjelmalla. Lopullinen rakenne ja ulkoasu tulisi selville vasta toteutusvaiheessa.



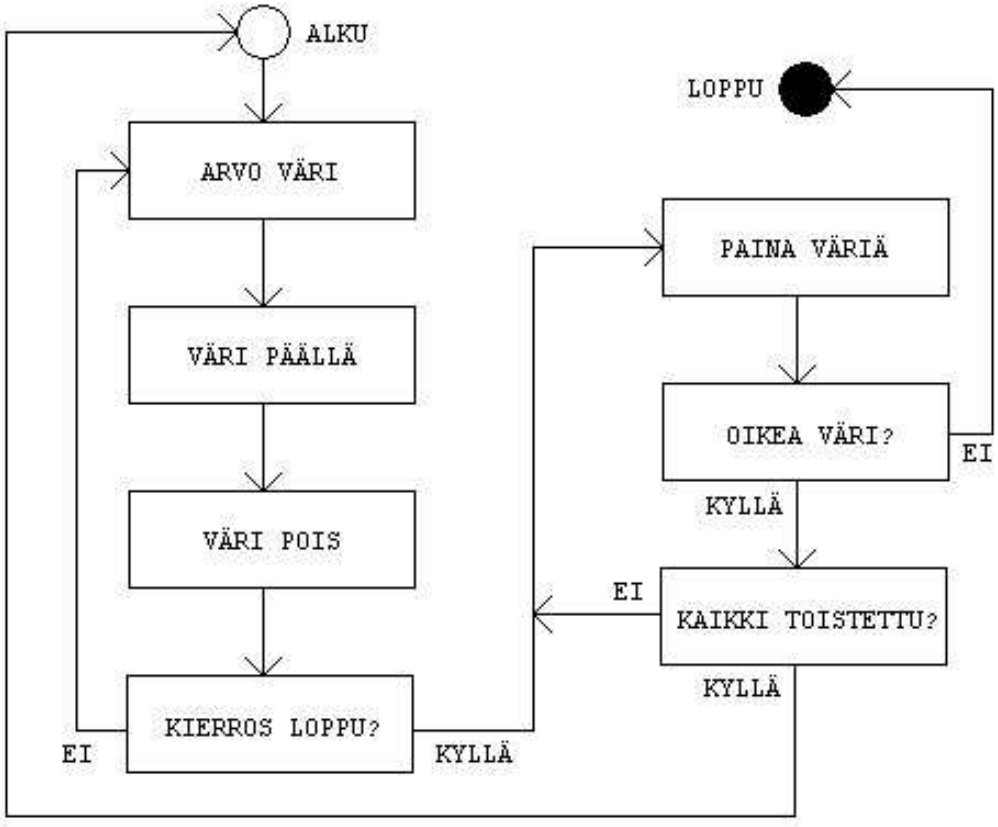
Kuva 14. Alkuperäinen luonnos sivustosta.

Koska sivun sisällyttäminen ei onnistunut SSI:llä, navigointivalikko on kopioitava jokaiseen sivuun erikseen. Näin yhden linkin lisääminen valikkoon olisi myös tehtävä jokaiseen sivuun. Tässä vaiheessa unohtunut ratkaisu tähän ongelmaan olisi voinut olla HTML-kielen sisältämä iframe. Iframe on HTML-elementti, joka sisältää toisen tiedoston sisällön. Sen käyttö on kuitenkin usein kyseenalaista, eikä HTML:n Strict-määrittely tue sitä lainkaan.

4.2 Pelien suunnittelu

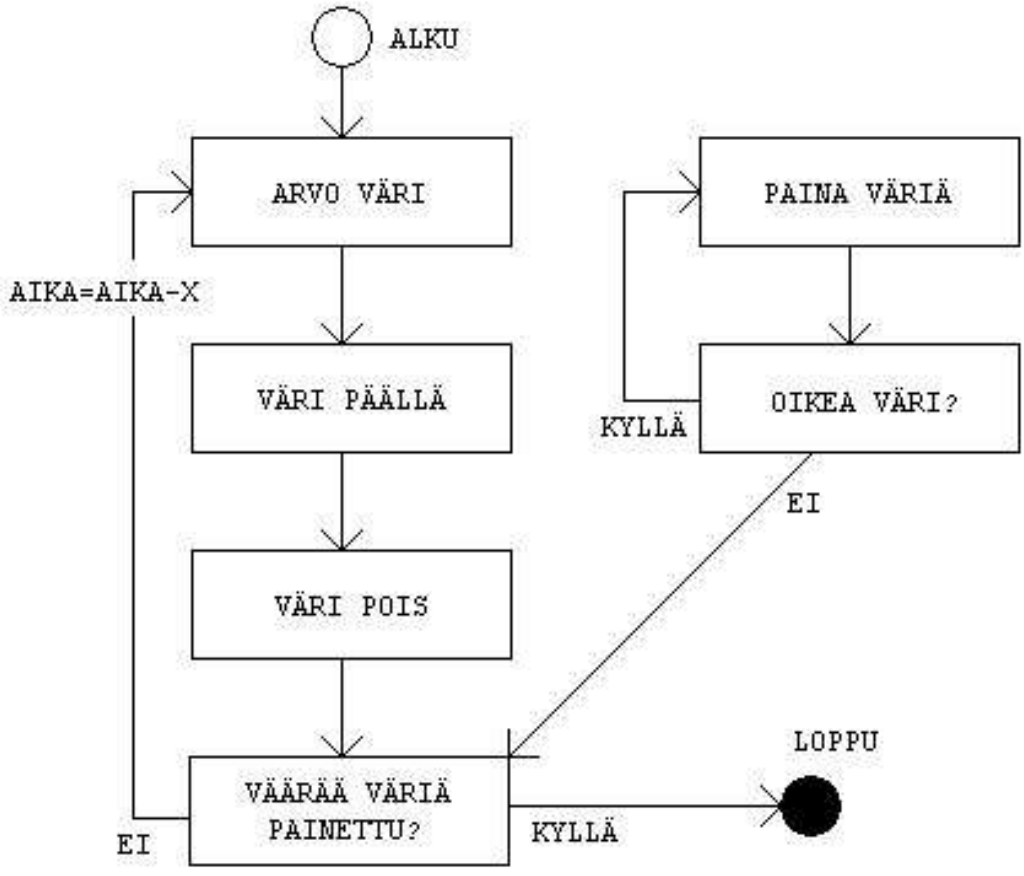
Työn määrittelyssä ei mainittu peleissä käytettäviä värejä, mutta käyttöön otettiin luonnollisesti punainen, vihreä, sininen ja keltainen. Värien esitystavalle oli vaihtoehtoina käyttää kuvia tai HTML-elementtejä, joiden taustaväriä muutetaan ohjelmallisesti. Kuvien käyttö oli ensisijainen valinta.

Muistipelissä yksi neljästä väristä syttyy kerrallaan palamaan niin monta kertaa, kuin sen hetkinen vaikeusaste edellyttää. Tämän jälkeen käyttäjä toistaa saman sarjan klikkaamalla aina oikeaa väriä. Kun kaikki värit on toistettu oikeassa järjestyksessä, alkaa seuraava kierros, jossa on yksi väri enemmän. On erittäin tärkeää, että käyttäjä voi alkaa toistamaan värejä vasta, kun koko sarja on ensin näytetty. Muuten värejä voisi painella sitä mukaa, kun ne syttyvät, eikä niitä tarvitsisi muistaa. Väärää väriä painettaessa peli päättyy ja sen voi aloittaa alusta.



Kuva 15. Muistipelin toimintaperiaate. Mikäli käyttäjä osaa toistaa koko sarjan, aloitetaan uusi sarja, jossa on yksi väri enemmän. Tasomäärä ei ole rajoitettu.

Nopeuspelissä värejä arvotaan ja näytetään jatkuvasti kiihtyvällä vauhdilla, ja tätä jatketaan niin kauan, kuin käyttäjä painaa väärää väriä. Toisin kuin muistipelissä, nopeuspelissä värejä on pakko saada painaa samaan aikaan, kun ne syttyvät. Käyttäjän painallukset ja värien vilkuttaminen tapahtuvat siis rinnakkain samanaikaisesti. Kun väärää väriä painetaan, peli loppuu ja sen voi aloittaa uudelleen. Värivaihdon nopeutuksen algoritmia oli tässä vaiheessa vaikea pohtia, joten se syntyi vasta toteutusvaiheessa tee ja testaa menetelmällä. Tässä vaiheessa on vaikea edes suurin piirtein sanoa, mikä on se värien vaihtoaika, jolloin käyttäjä ei enää pysty toistamaan värejä. Vaikka silmä pystyisikin seuraamaan peliä vielä, saattaa hiirellä klikkailu niin nopeasti olla liian vaikeaa.



Kuva 16. Nopeuspelin toimintaperiaate. Jokaisen värin jälkeen värin näyttöaika vähenee.

5 WEB-SIVUJEN TOTEUTUS

5.1 Rakenne ja ulkoasu

Ensimmäiseksi peleille syntyi väliaikainen, yksinkertainen käyttöliittymä, jotta niiden ohjelmointia pystyi tekemään samanaikaisesti muun käyttöliittymän kanssa. Kuvilla esitettävät värit oli helppo tehdä Paint Shop Pro 5-ohjelmalla, jolla kaikki työssä käytetyt kuvat tehtiin. Tämän jälkeen alkoi sivun ulkoasun rakentaminen suunnitelman mukaisesti.

Yläbannerin logon tekemiseen meni jonkin verran aikaa, koska siihen tuli kokeiltua paljon eri vaihtoehtoja. Lopulta yläreunaan tuli mustaan viiva, joka on alkaessaan hieman paksumpi ja kapenee porrastetusti ohuemmaksi. Banneriin tuli myös E1101 teksti. Viivaan ja tekstiin lisättiin valkoiset varjostukset, jotta ne näyttäisivät hienommilta.

Kuva 17. Sivuston banneri.

Sivuston rakenteen ja tyylien tekeminen ei ollut mitenkään hankalaa. Pitkäaikaisen HTML ja CSS kielien käytön ansiosta niiden käyttö sujui täysin rutiininomaisesti. Aikaa kului kuitenkin yksityiskohtien tekemiseen. Jokaisen osan kohdalla tuli koitettua montaa eri vaihtoehtoa, ennen kuin sopiva löytyi. Sivustolle tuli aluksi kolme sivua: etusivu, muistipelisivu ja nopeuspelisivu. Tämä tarkoittaa kolmea HTML-tiedostoa, joissa jokaisessa oli sivun koko ulkoasu toteutettuna. Myöhemmin lisäyksenä tuli vielä neljäs diagnostiikka-sivu, joka näyttää diagnostiikkatietoa operointipaneelista. Diagnostiikkatietojen haku onnistui operointipaneelin valmiilla scriptillä.

Toisen sivun sisällyttäminen sivuun ei ollut mahdollista SSI:llä. Koska Ajaxilla pystyy lähettämään HTTP-pyynnön, sisällytys on sen avulla mahdollista. Tarvitsee vain ajaa heti sivun latautuessa JavaScript-funktio, joka lähettää Ajax-pyynnön. Käytännössä näin ollen sivua palvelimelta haettaessa, palvelimelle lähetetään kaksi pyyntöä. Sivun latauksen yhteyteen haluttavan funktion ajon voi sijoittaa HTML:ssä body-tagin onload attribuutiksi. Tällä taktiikalla onnistui navigointivalikon sisällytys jokaiselle sivulle, jolloin esimerkiksi uuden linkin lisääminen tarvitsee tehdä vain valikkotiedostoon ja se näkyy automaattisesti kaikilla sivuilla. Koska Ajax on selainriippuvainen tekniikka, HTTP-pyynnön lähetys ei ole sama jokaiselle selaimelle. Tähän löytyi valmis funktio, jossa yleisimmät selaimet on otettu huomioon.

```
function getFile(file,element) {
    var xmlHttp;
    try {
      // Firefox, Opera 8.0+, Safari
      xmlHttp=new XMLHttpRequest();
    } catch (e) {
      // Internet Explorer
      try {
        xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
      } catch (e) {
        try {
          xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e) {
          alert("Selain ei tue AJAXIA!");
          return false;
        }
      }
    }
    xmlHttp.onreadystatechange=function() {
        if(xmlHttp.readyState==4){
          document.getElementById(element).innerHTML=xmlHttp.responseText;
        }
    }
    xmlHttp.open("GET",file,true);
    xmlHttp.send(null);
}
```

Kuva 18. Ajax-kutsun suorittava JavaScript-funktio, joka toimii monissa eri selaimissa. Parametreina annetaan haettavan tiedoston nimi ja sen elementin ID, johon vastauksena tuleva sisältö laitetaan.

```
<body onload="getFile('valikko.html','vasen')">
...
<td id="vasen">
...
```

Kuva 19. Ajax-kutsun ajaminen heti sivun latauksen yhteydessä onnistui näin. Tiedoston valikko.html sisältö sijoitetaan td-tagin sisään, jonka id on vasen.

Ratkaisu ei ollut kuitenkaan täysin miellyttävä, koska koko näkymä on joka sivulla samanlainen ja vain pääosan sisältö itse asiassa vaihtuu. Hetken mietinnän ja kokeilun jälkeen toteutus muuttui sellaiseksi, että koko ulkoasu bannereineen ja valikoineen on vain yhdessä tiedostossa (index.html). Sisältöosa tässä tiedostossa on tyhjä, mutta heti sivun latautuessa siihen haetaan oletussivu, jonka nimeksi tuli etusivu.html.

Valikon linkkejä painamalla aina sama funktio hakee ensimmäisenä parametrina annetun sivun sisällön toisena parametrina annetun elementin sisällöksi. Näin ollen sivu ladataan operointipaneelista kokonaisuudessaan vain kerran, eikä sivua vaihdettaessa tarvitse kuin hakea pelkkä vaihdettava sisältö. Tämä oli erinomainen ratkaisu kaikin puolin, koska vain muuttuva tieto vaihtuu. Haettavissa sisältösivuissa ei siis ollut enää html, head, body tai muita vastaavia HTML-tageja.

Sivujen lopullinen ulkoasu on liitteenä (Liite 1).

5.2 Pelien toteutus

Pelien ohjelmointi alkoi etsimällä JavaScript-funktiota, jolla ohjelman saa pysäytettyä tietyksi aikaa. Tietoa löytyi internetistä, joka on pullollaan aiheeseen liittyviä oppaita. Aluksi löytyi useita muiden itse tekemiä funktioita, jotka perustuivat siihen, että nykyinen ajanhetki tallennetaan muuttujaan. Tämän jälkeen käydään silmukkaa läpi niin kauan, että nykyhetken ja tallennetun hetken ero on yli halutun odotusajan.

```
function pausecomp(millis)
{
var date = new Date();
var curDate = null;

do { curDate = new Date(); }
while(curDate-date < millis);
}
```

Esimerkki löytämästäni odotusfunktiosta. Haluttu aika annetaan millisekunteina.

Tällaisen funktion odotusaika toimi kuitenkin vain kerran, jonka jälkeisillä odotuksilla ei enää ollut merkitystä. Lisäetsinnöillä löytyi nopeasti toinen vaihtoehto: JavaScriptin valmis funktio, jonka avulla ohjelman saa odottamaan ennen, kuin se siirtyy eteenpäin. Tämä funktio on setTimeout, jolle annetaan kaksi parametria. Ensimmäinen parametri on koodi, joka halutaan suorittaa toisena parametrina annetun ajan jälkeen. Aika annetaan millisekunteina. Tätäkään funktiota ei kuitenkaan voinut käyttää kahta kertaa peräkkäin, jolloin sarja ajastettuja tapahtumia olisi mahdollista muisti- ja nopeuspelien edellyttämällä tavalla.

```
setTimeout("alert('2 sekuntia')", 2000);
setTimeout("alert('toiset 2 sekuntia')", 2000);
```

Tämä koodi ei toiminut niin, että molemmat ilmoitukset olisivat tulleet kahden sekunnin välein, vaan molemmat ilmoitukset tulivat samaan aikaan kahden sekunnin jälkeen.

Pienen mietinnän jälkeen peräkkäisten odotusten ongelmaan löytyi ratkaisu, joka oli rekursiivisen eli itseään kutsuvan funktion käyttäminen. Rekursiivisen funktion avulla ohjelma toimi halutulla tavalla, eli sillä pystyi tekemään rajattoman määrän tapahtumia tietyin väliajoin.

```
function funktio() {
    if(i==0) {
        i=1;
    } else {
        i=0;
    }
    document.getElementById("h1").innerHTML = i;
    setTimeout("funktio()", 1000);
}
```

Kuva 20. Tämä funktio vaihtaa elementin, jonka id on h1, sisällä olevan tekstin 0:ksi ja 1:ksi vuorotellen yhden sekunnin välein.

Pelien toteutus onnistuu tällä periaatteella lisäämällä funktion sisälle ehtolauseita, muuttujien asetuksia ja funktion itsensä kutsuja. Ennen pelien ohjelmointia oli kuitenkin tarpeellista tehdä funktiot, joilla väri saadaan päälle ja pois päältä. Aluksi värit esitettiin kuvien avulla, jolloin vaihtofunktio vaihtoi kyseisen värin kuvaa mutta tämä ratkaisu vaihtui myöhemmin pelkän HTML-elementin käyttämiseen, jonka taustaväriä vaihdetaan. Kuvat olivat huono ratkaisu, koska ne ovat tiedostoja jotka pitää ladata, eikä vaihto ollut aina täysin huomaamaton. Välillä väri välähti häiritsevästi kuvien vaihtuessa. Pelkän HTML-elementin taustaväriä vaihtaen värien vaihto on paljon sujuvampi.

```
147  function turn_on(koodi) {
148   switch(koodi) {
149    case 1:
150      td = "red";
151      vari = "#FF0000";
152    break;
153    case 2:
154      td = "green";
155      vari = "#00FF00";
156    break;
157    case 3:
158      td = "blue";
159      vari = "#0000FF";
160    break;
161    case 4:
162      td = "yellow"
163      vari = "#FFFF00";
164    break;
165   }
166   document.getElementById(td).style.background = vari;
167  }
```

Kuva 21. Värin sytyttäminen.

```
171  function turn_off(koodi) {
172   var tdcolor = new Array();
173   switch(koodi) {
174    case 1:
175      td = "red";
176      vari = "#500000";
177    break;
178    case 2:
179      td = "green";
180      vari = "#005000";
181    break;
182    case 3:
183      td = "blue";
184      vari = "#000050";
185    break;
186    case 4:
187      td = "yellow"
188      vari = "#505000";
189    break;
190   }
191   document.getElementById(td).style.background = vari;
192  }
```

Kuva 22. Värin sammuttaminen.

5.2.1 Muistipeli

Pelien teko oli lopulta melko helppoa, eikä mennyt kauan aikaa siihen, että muistipelin sai sytyttämään värejä haluttu määrä kerrallaan. Koska funktio kutsuu itseään, joidenkin muuttujien arvoja piti tarkastella ennen, kuin niitä on asetettu. Tällaisiin muuttujiin piti lisätä ehtolauseella tarkastus siitä, onko muuttuja olemassa JavaScriptin funktiolla typeof(). Tämä johtuu siitä, että muuttujan käyttö ennen kuin se on asetettu johtaa koko funktion toimimattomuuteen. Tai jos funktion alussa piti alustaa jokin muuttuja, mutta vain pelin alkaessa tarvitsi sen olemassaolo myös tarkistaa samalla funktiolla.

Aina, kun väri sytytetään muuttuja ”palauta” saa arvon tosi (true). Tämän jälkeen funktiota kutsutaan pienellä viiveellä, jonka ajan väri on päällä. Kun aika on mennyt, funktio suoritetaan taas ja siinä tarkistetaan, onko ”palauta” tosi. Jos se on tosi, väri sammutetaan, ”palauta” asetetaan epätodeksi (false) ja funktiota kutsutaan taas pienellä viiveellä. Tämä viive on aika, jonka väri on pois päältä. Nyt ”palauta” on epätosi, joten uusi väri arvotaan ja se sytytään, jolloin ”palauta” asetetaan jälleen todeksi ja funktiota kutsutaan. Tätä jatketaan aina tason edellyttämän määrän ajan. Pelin funktiota kutsutaan parametrilla taso, joka kertoo minkä verran värejä näytetään. Jokaisen värin näyttämisen jälkeen funktio kutsuu itseään vähentäen tasosta yhden. Kun taso on 0, funktion ajo lopetetaan. Lähtötilanteessa funktiota kutsutaan siis parametrilla 1.

Tasojen välissä käyttäjän pitää toistaa sarja, joten tätä varten tarvittiin funktio, joka tarkistaa painallukset. Käytännössä koko muistipelin suoritus lopetetaan jokaisen tason välissä. Ainoastaan yksi muuttuja säilyy, joka on nimeltään ”sarja”. Tässä muuttujassa on kaikki tason näyttämät värit tallennettuna järjestyksessä. Tarkistusfunktio ei saa tehdä mitään, mikäli peli ei ole lainkaan käynnissä tai värejä juuri näytetään. Tämä tapahtuu tarkistamalla, että sarja-muuttuja on olemassa, mutta pelin aikana käytetty ”vaihdot”-muuttuja ei ole. Tämän jälkeen funktio tarkistaa, montako kertaa funktiota on jo kutsuttu (jotakin väriä klikattu) ja sen perusteella vertailee painettua väriä sarja-muuttujan niin monenteen arvoon.

Kun värejä on toistettu sarja-muuttujan sisältämän arvomäärän verran, tarkistusfunktio kutsuu muistipelifunktiota arvolla sarja-muuttujan arvomäärä+1 ja poistaa sarja-muuttujan sekä muuttujan, joka sisälsi käyttäjän painamien värien määrän. Näin seuraava taso aloittaa taas uuden sarjan tyhjästä ja sen jälkeinen käyttäjän klikkaamien värien määrä aloitetaan myös tyhjästä, jotta sitä osataan verrata oikeaan sarja-muuttujan indeksiin. Väärää väriä klikattaessa muuttujat tuhotaan, eikä muistipeliä enää kutsuta. Tämän jälkeen pelin voi aloittaa uudelleen alusta tasolta 1.

Pelin voi aloittaa sivulla olevasta lomakenapista. Aloitusnappia ei kuitenkaan pitäisi voida painaa silloin, kun peli on jo käynnissä. Tämä sekoitti pelin täysin, koska muuttujilla oli jo kesken olevan pelin vuoksi arvoja. Ensimmäisenä ratkaisuna koko nappi kadotettiin sivulta, kun peli alkaa. Tämä oli kuitenkin huono ratkaisu, koska napin palautusvaiheessa sen luontiin tarvittava koodi piti kirjoittaa kokonaan uudelleen JavaScript-funktiossa. Tämän vuoksi parempi ratkaisu oli pitää nappi koko ajan näkyvillä, mutta estää sen painaminen pelin aikana. Tämä oli myös paljon helpompi ratkaisu. Painalluksen estäminen onnistuu JavaScriptin avulla asettamalla nappi-elementin ominaisuus ”disabled” arvoon 1.

Värin välähdys sitä klikattaessa tuntui hyvältä lisäykseltä, koska siten käyttäjä saa väriä painaessaan varmistuksen klikkauksestaan eikä jää epäselväksi, että ottiko peli painalluksen vai ei. Tämä onnistui lisäämällä värin sytytysfunktio värin näyttävän elementin onclick-ominaisuuteen tarkistusfunktion lisäksi sekä sammuttamisfunktio pienellä viiveellä. Pelin yhteyteen tuli myös tason ja värilaskurin arvot näkyville. Värilaskuri laskee pelin näyttämiä värejä.

Aika, jonka värit ovat päällä ja poissa oli aluksi yksi sekunti. Myöhemmin tämä aika vaihtui kuitenkin 800 millisekuntiin, joka oli sopivampi. Värin välähtämisajaksi sitä klikattaessa tuli 300 millisekuntia, jolloin se välähtää vain nopeasti.

Peli oli melko nopeasti valmis, mutta sen jälkeiseen hienosäätöön ja korjaukseen menikin aikaa moninkertaisesti. Tämän vuoksi muistipelin testaus tulikin katettua melko laajasti jo sen ohjelmoinnin aikana.

Jokaisen värin sammuttamisen jälkeen ohjelma odottaa asetun ajan ja tämä vaikuttaa tietysti myös viimeiseen väriin. Tämä seikka aiheutti harhaluulon siitä että peli toimii väärin. Jos käyttäjä alkoi toistaa sarjaa heti viimeisen värin sammuttua, peli ilmoitti usein painalluksen olevan väärin vaikka se ei ollut. Ensimmäinen klikkaus osui tietysti viimeisen värin odotusajalle, eikä sitä vielä laskettu painallukseksi. Näin ollen käyttäjän luullessa toistavansa jo toista väriä, muistipeli tulkitsi sen vasta ensimmäiseksi väriksi. Tämän vuoksi painettu väri oli tietysti yleensä väärin ja virheen ymmärtäminen vei jonkin verran aikaa, koska painallus oli välillä kuitenkin sattumalta oikein. Vikaa tuli turhaan etsittyä tarkastusfunktiosta ja muusta koodista. Todellisuudessa mitään vikaa ei ollut, koska peli oli ohjelmoitu toimimaan juuri näin. Ongelma kuitenkin löytyi ja viimeisen värin jälkeinen odotusaika oli otettava pois.

Tämän kaltaisia pieniä niin sanottuja bugeja pelistä löytyi jonkin verran. Niiden huomaaminen saattoi olla todella vaikeaa ja niiden korjaamiseen kului paljon aikaa. Bugien aiheuttajat löytyivät usein loogisen toiminnan teoreettisella läpikäymisellä, joten muistipelin tekemisessä aikaa meni ajattelemiseen kymmeniä kertoja enemmän, kuin koodaukseen.

```
42  function tarkista(id) {
43   if(typeof(vaihdot) == "undefined" && typeof(sarja) != "undefined") {
44    if(typeof(painettu) == "undefined") painettu=1; else painettu++;
45    if(sarja[painettu-1] != id) {
46     document.getElementById("laskuri").innerHTML = "0";
47     document.getElementById("virhe").innerHTML = "Painoit väärää väriä!";
48     document.getElementById("pelaa").disabled = 0;
49     delete painettu;
50     delete sarja;
51     pisteet = document.getElementById("taso").innerHTML;
52     send = '<form action="http://www.palvelin.fi/peli.php" method="post">';
53     send += 'Nimimerkki: <input type="text" name="nimi" maxlength="12" />';
54     send += '<input type="hidden" name="peli" value="muistipeli" />';
55     send += '<input type="hidden" name="pisteet" value="'+pisteet+'" />';
56     send += '<input type="submit" id="pelaa" value="Tallenna" />';
57     send += '</form>';
58     document.getElementById("sendscore").innerHTML = send;
59    }
60    else if(sarja.length == painettu) {
61     setTimeout("muistipeli("+(sarja.length+1)+")",1000);
62     delete painettu;
63     delete sarja;
64    }
65   }
66  }
```

Kuva 23. Käyttäjän klikkauksen oikeellisuuden tarkistaminen muistipelissä.

```
function muistipeli(taso) {
 if(typeof(sarja) == "undefined") sarja = new Array(taso);
 if(typeof(vaihdot) == "undefined") vaihdot = 0;
 if(typeof(palauta) == "undefined") palauta = true;
 if(palauta && typeof(numero) == "undefined") {
  document.getElementById("sendscore").innerHTML = "";
  document.getElementById("pelaa").disabled = 1;
  document.getElementById("virhe").innerHTML = "";
  document.getElementById("taso").innerHTML = taso;
  document.getElementById("laskuri").innerHTML = "0";
  palauta = false;
  setTimeout("muistipeli("+(taso)+")",800);
 }
 else if(palauta) {
  turn_off(numero);
  palauta = false;
  if((taso+vaihdot) == vaihdot) {
   muistipeli(taso);
  }
  else {
   setTimeout("muistipeli("+(taso)+")",800);
  }
 }
 else if(taso>0) {
  palauta=true;
  numero = Math.floor(Math.random()*4)+1;
  sarja[vaihdot] = numero;
  turn_on(numero);
  vaihdot++;
  document.getElementById("laskuri").innerHTML = vaihdot;
  setTimeout("muistipeli("+(taso-1)+")",800);
 } else {
  turn_off(numero);
  delete vaihdot;
  delete numero;
  delete palauta;
 }
}
```

Kuva 24. Muistipelin koodi.

5.2.2 Nopeuspeli

Muistipeli antoi todella hyvän pohjan nopeuspeliin. Nopeuselin toimintaperiaate on kuitenkin melko erilainen. Muistipelissä värejä esitetään sarja kerrallaan ja välillä odotetaan käyttäjän toistamista, jonka jälkeen taas aloitetaan uuden sarjan näyttö. Nopeuspelin toiminta ei taas riipu mitenkään käyttäjästä. Pelissä tarvitsee vain vaihtaa värejä kiihtyvällä vauhdilla ja tallentaa toteutettu sarja, jotta käyttäjän painallukset voidaan tarkistaa.

Nopeuspeliä kutsutaan parametrilla aika, joka kertoo kuinka kauan väriä pidetään päällä. Jokaisen värin jälkeen, funktion kutsuessa itseään, aikaa vähennetään. Nopeuspelin alkaessa aika on tuhat millisekuntia eli yksi sekunti, josta se lähtee vähenemään. Ajan vähentämiseen ei löytynyt yhtä sopivaa algoritmia, jolla peli toimisi sopivalla tahdilla. Tämän vuoksi toteutus tuli tehtyä tarkistamalla aika ehtolauseiden avulla ja muuttamalla sitä tarvittava määrä sen perusteella. Alussa aika vähenee nopeammin ja lopussa hitaammin, eli sen kiihtyvyys on logaritminen. Ensimmäinen vähennys on 1000 millisekunnista 900 millisekuntiin, mutta lopussa ajasta vähennetään vain yksi millisekunti.

Nopeuspeliin oli myös tehtävä tarkistusfunktio, joka tarkistaa käyttäjän painamat värit. Toisin kuin muistipelissä, nopeuspelissä värejä klikataan pelin ollessa käynnissä. Käyttäjän ei tarvitse pysyä pelin tahdissa, vaan muistipelin tarkistuksen tapaan klikkausten määrä lasketaan ja sen perusteella tehdään vertaus sarjan oikeaan indeksiin. Tosin värejä on mahdoton muistaa suurilla nopeuksilla, joten käytännössä käyttäjän on pysyttävä pelin tahdissa kokoajan. Tarkistusfunktio tarkistaa ensin onko peli käynnissä ja jos on, tarkistus suoritetaan. Käyttäjän klikatessa väärää väriä muuttuja ”vaihdot”, jossa on tallessa näytettyjen värien määrä, asetetaan arvoon ”stop”. Nopeuspelin seuraavalla kierroksella tämä muuttujan arvo johtaa pelin lopetukseen, eikä värejä enää näytetä.

Nopeuspelin testauksessa tuli ilmi vaihtonopeudesta johtuvia, suuria käytännön eroja muistipeliin verrattuna. Muistipelissä väri vilahtaa klikkauksen varmistukseksi, kun käyttäjä painaa sitä. Tätä toimintoa ei pidä missään nimessä toteuttaa nopeuspeliin. Kun värit vilkkuvat yhä nopeammin ja nopeammin nopeuden kasvaessa, värin vilahtaminen sitä painettaessa aiheuttaa ainoastaan suurta sekaannusta, eikä käyttäjä välttämättä tiedä mikä väri oikeasti vilkahti.

Toisena asiana muistipelissä väri sammuu joka välissä. Nopeuspelissä myös tämä on huono asia, joka saattaa sekoittaa käyttäjän. Pelin toiminta piti muuttaa sellaiseksi, että jokin väri on koko ajan päällä. Eli, kun edellinen väri sammuu, niin samaan aikaan jo seuraava syttyy, eikä välissä ole taukoa.

Kolmas nopeudesta johtuva seikka oli saman värin toistaminen kaksi kertaa peräkkäin. Jo vähänkin suuremmilla nopeuksilla alkaa olla todella vaikeaa tietää, kuinka monta kertaa peräkkäin väri syttyi. Se on suorastaan mahdotonta, kun edellisen muutoksen jälkeen väri ei edes sammu välissä. Tämän vuoksi koodia oli muutettava niin, että nopeuspelissä sama väri ei voi tulla kahta kertaa peräkkäin.

Nopeuspeli vaihtoi värejä alun perin niin kauan, että aika meni nollaan. Tämä oli kuitenkin turhaa, koska peli ei käytännössä koskaan etene niin pitkälle, ellei sitä vain jätetä päälle eikä sarjaa edes yritetä toistaa. Lopullisessa versiossa nopeuspeli päättyy 300 värivaihdon jälkeen, jolloin värin vaihtoaika on 89 millisekuntia. Tähän tulokseen on ihmisen täysin mahdoton päästä, joten pelin ei ole tarve jatkaa siitä eteenpäin. Käyttöliittymään tuli näkyviin oikein painettujen värien määrä, näytettyjen värien määrä ja värien vaihtoaika millisekunteina.

Myös muistipelissä oli monia pieniä virheitä. Niiden korjaus vei samoin kuin muistipelissä paljon enemmän aikaa, kuin itse pelin tekeminen. Koska uuden pelin voi aloittaa heti edellisen loputtua, eikä sivua ladata välissä uudelleen, oli JavaScript-muuttujien alustuksien kanssa oltava erittäin tarkkana. Nopeuspelin koodista tuli sen eroavaisuuksien vuoksi hieman muistipelin koodia pidempi, vaikka nopeuspeli oli itse asiassa muistipeliä yksinkertaisempi tehdä.

```
70  function nopeuspeli(aika) {
71   document.getElementById("vaihto").innerHTML = Math.floor(aika);
72   if(typeof(sarja) == "undefined") sarja = new Array();
73   if(typeof(vaihdot) == "undefined") vaihdot = 0;
74   if(typeof(palauta) == "undefined") palauta = true;
75   if(palauta && typeof(numero) == "undefined") {
76    document.getElementById("pelaa").disabled = 1;
77    document.getElementById("sendscore").innerHTML = "";
78    document.getElementById("virhe").innerHTML = "";
79    document.getElementById("taso").innerHTML = "0";
80    document.getElementById("laskuri").innerHTML = "0";
81    palauta = false;
82    setTimeout("nopeuspeli("+aika+")",aika);
83   }
84   else if(palauta) {
85    palauta = false;
86    if(aika>=800) uusiaika = (aika-aika/10);
87    else if(aika>=700) uusiaika = (aika-aika/25);
88    else if(aika>=600) uusiaika = (aika-aika/50);
89    else if(aika>=500) uusiaika = (aika-aika/75);
90    else if(aika>=400) uusiaika = (aika-aika/100);
91    else if(aika>=300) uusiaika = (aika-aika/150);
92    else uusiaika = (aika-1);
93    nopeuspeli(uusiaika);
94   }
95   else if(aika>0 && vaihdot != "stop" && vaihdot < 300) {
96    if(typeof(numero) != "undefined") turn_off(numero);
97    numero = Math.floor(Math.random()*4)+1;
98    if(numero == sarja[vaihdot-1]) {
99     nopeuspeli(aika);
100   }
101   else {
102    palauta=true;
103    sarja[vaihdot] = numero;
104    turn_on(numero);
105    vaihdot++;
106    document.getElementById("laskuri").innerHTML = vaihdot;
107    setTimeout("nopeuspeli("+aika+")",aika);
108   }
109  } else {
110   if(typeof(numero) != "undefined") turn_off(numero);
111   delete numero;
112   delete palauta;
113   if(vaihdot == "stop") delete vaihdot;
114  }
115 }
```

Kuva 25. Nopeuspelin koodi.

```
119 function tarkistanopeus(id) {
120  if(typeof(vaihdot) != "undefined" && typeof(sarja) != "undefined") {
121   if(typeof(painettu) == "undefined") painettu=1; else painettu++;
122   if(sarja[painettu-1] != id) {
123    document.getElementById("virhe").innerHTML = "Painoit väärää väriä!";
124    if(typeof(lopeta) == "undefined") lopeta = 1;
125    document.getElementById("pelaa").disabled = 0;
126    delete painettu;
127    delete sarja;
128    vaihdot = "stop";
129    if(typeof(numero) == "undefined") delete vaihdot;
130    pisteet = document.getElementById("taso").innerHTML;
131    send = '<form action="http://www.palvelin.fi/peli.php" method="post">';
132    send += 'Nimimerkki: <input type="text" name="nimi" maxlength="12" />';
133    send += '<input type="hidden" name="peli" value="nopeuspeli" />';
134    send += '<input type="hidden" name="pisteet" value="'+pisteet+'" />';
135    send += '<input type="submit" id="pelaa" value="Tallenna" />';
136    send += '</form>';
137    document.getElementById("sendscore").innerHTML = send;
138   }
139   else {
140    document.getElementById("taso").innerHTML = painettu;
141   }
142  }
143 }
```

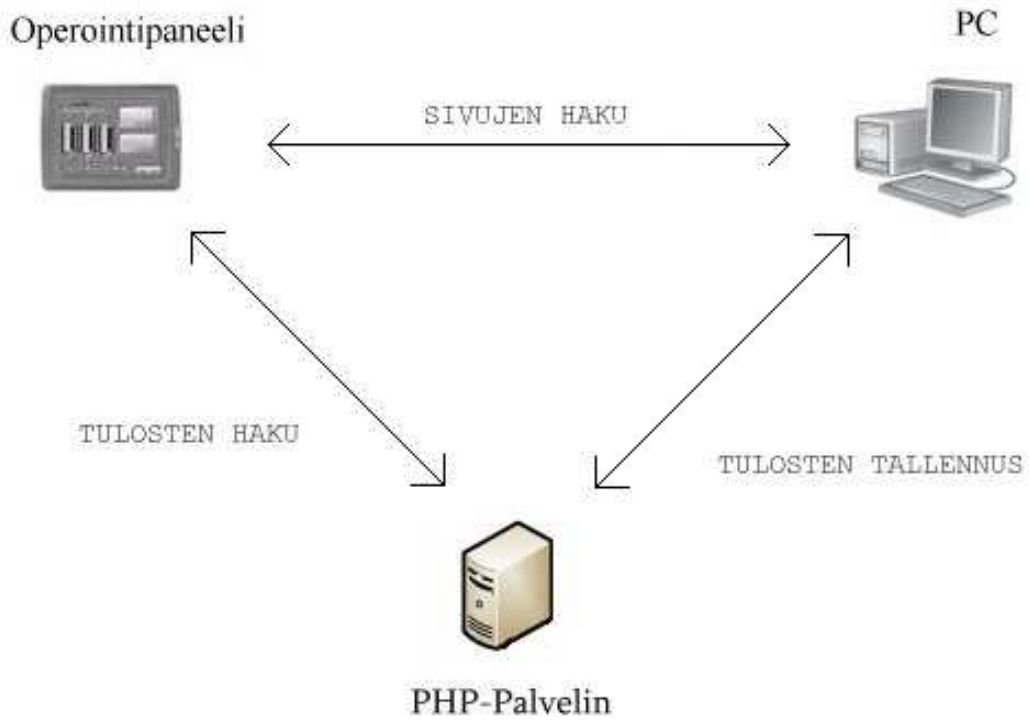
Kuva 26. Käyttäjän klikkauksen oikeellisuuden tarkistaminen nopeuspelissä.

5.3 Tulosten tallennus

Tulosten tallennus operointipaneelin www-palvelimelle ei ollut mahdollista, kuten esiselvityksessä tuli todettua. Tämän vuoksi tallentamisen toteutus oli vähällä jäädä kokonaan pois. Myöhemmin tallennus päätettiin kuitenkin toteuttaa, mikäli se vain on millään tavalla mahdollista. Monia ratkaisuja tuli mietittyä, kuten mahdollisuutta käyttää joitakin operointipaneelissa valmiina olevia muuttujia, joita pystyy asettamaan valmiiden scriptien avulla. Tämä oli kuitenkin todella kaukaa haettua, eikä se ollut missään nimessä toteutettava ratkaisu. Lisäksi tulosten pariksi pitää saada jokin nimimerkki, jotta tiedetään kuka tuloksen on saanut tai koko tallentamisesta ei ole mitään hyötyä. Yksinkertaisimmillaan nimimerkki ja tulos voisivat olla tekstitiedostossa jollakin välimerkillä erotettuna ja niitä olisi allekkain niin monta, kuin tuloksia halutaan säilyvän.

Lopulta oli päädyttävä käyttämään ulkoista palvelinta tulosten tallentamiseen, koska vaihtoehtoja ei juuri ollut. Tämä oli tässä tilanteessa pakollista. Tallennukseen käytettäväksi ohjelmointikieleksi tuli PHP (PHP: Hypertext Preprocessor). PHP on komentosarjakieli, joka on tämän hetken johtava dynaamisten web-palveluiden tuottamiseen tarkoitettu ohjelmointikieli. Siinä on erittäin laajat ja monipuoliset merkkijonojen käsittelyfunktiot, joiden avulla tulosten käsittely ja tallennus on todella helppo toteuttaa.

Tallennuksen lisäksi tiedot pitäisi pystyä lukemaan operointipaneelin sivulle. Tulosten lukeminen suoraan tekstitiedostosta operointipaneelin sivuille ei ole mahdollista. Ratkaisuna tähän ongelmaan oli se, että tiedot tallennetaan JavaScript-syntaksin mukaisina muuttujina JavaScript-tiedostoon, jonka voi sisällyttää HTML-sivulle <script > tagilla ja muuttujat saadaan käyttöön.



Kuva 27. Tulosten tallennus tapahtui erilliselle palvelimelle, josta operointipaneeli haki ne JavaScript-tiedostona, ennen sivujen lähetystä käyttäjälle. Tallennus tapahtui suoraan käyttäjältä PHP-palvelimelle.

Käytännön toteutus oli sellainen, että käyttäjä ei edes tiedä, että käytössä on erillinen palvelin tulosten tallennukseen. Peleihin piti lisätä koodi, jonka avulla pelien loputtua sivulle ilmestyy tulosten lähetyslomake. Lomakkeessa on syöttökenttä käyttäjän nimimerkille ja tallennusnappi. Pelin tulos sisällytetään lomakkeeseen piilotettuna kenttänä. Tallenna-nappia painamalla lomake lähetetään PHP-palvelimelle. Tulosta ei tarvitse yrittää tallentaa, vaan käyttäjä voi myös aloittaa uuden pelin, jolloin lomake katoaa sivulta.

PHP-palvelimella PHP-scripti lukee ensin tiedostosta vanhat tulokset talteen ja avaa sen jälkeen kyseisen tiedoston kirjoitusta varten tyhjentäen sen samalla. Tämän jälkeen scripti kirjoittaa tulokset tiedostoon uudelleen sisällyttäen lähetetyn tuloksen oikeaan paikkaan. Scripti pitää tiedostossa tallennettuna kymmenen parasta tulosta. Koska molemmissa peleissä tulosten tallennus oli täysin samanlainen, saman scriptin käyttö sopi molempien pelien tulosten tallennukseen. Lähetyslomakkeeseen oli vain lisättävä uusi piilotettu kenttä joka kertoo, kumman pelin tulos on kyseessä.

```
1   muistipisteet = new Array(
2   "Tyhjä|0",
3   "Tyhjä|0",
4   "Tyhjä|0",
5   "Tyhjä|0",
6   "Tyhjä|0",
7   "Tyhjä|0",
8   "Tyhjä|0",
9   "Tyhjä|0",
10  "Tyhjä|0",
11  "Tyhjä|0"
12  );
```

Kuva 28. Tulosten tallennusformaatti on tämänkaltainen. Nopeuspelissä taulukon nimi on nopeuspisteet. Muistipelin pisteet tallennetaan muistipeli.js-tiedostoon ja nopeuspelin nopeuspeli.js-tiedostoon. Nimimerkki ja tulos erotetaan pystyviivalla.

Muistipelissä pistemäärä on viimeisin läpäisty taso ja nopeuspelissä oikein klikattujen värien määrä. Koska lomakkeella lähetetty muistipelin tulos on taso jossa peli päättyi eikä tasoa läpäisty, PHP-scriptissä muistipelin pisteistä oli vähennettävä yksi, jotta saadaan viimeisen läpäistyn tason tulos. Tulosten tallennuksen jälkeen scripti ohjaa käyttäjän takaisin operointipaneelin sivulle PHP:n funktiolla header(). Kaikki tämä tapahtuu niin nopeasti, ettei käyttäjä huomaa käyneensä ulkoisella PHP-palvelimella.

```php
<?php
if($_SERVER['REQUEST_METHOD'] == "POST") {

$peli = $_POST['peli'];
$pisteet = $_POST['pisteet'];
if($peli == "muistipeli") $pisteet--;
$nimi = str_replace("|", "", htmlspecialchars($_POST['nimi']));

$vanhat_tulokset = file($peli.".js");
$ekarivi = array_shift($vanhat_tulokset);
$vikarivi = array_pop($vanhat_tulokset);

for($i=0;$i<count($vanhat_tulokset);$i++) {
    $rivi = explode("\"",$vanhat_tulokset[$i]);
    $info = explode("|",$rivi[1]);
    if($pisteet > $info[1] && $saved == false) {
        $uudet_tulokset[] = "\"".$nimi."|".$pisteet."\"";
        $saved = true;
    }
    $uudet_tulokset[] = "\"".$info[0]."|".$info[1]."\"";
}

$fo = fopen($peli.".js", "w");
fputs($fo, $ekarivi);
for($i=0;$i<10;$i++) {
    if($i<9) $colon = ",\n"; else $colon = "\n";
    fputs($fo, $uudet_tulokset[$i].$colon);
}
fputs($fo, $vikarivi);
fclose($fo);

}
header("Location: ".$_SERVER['HTTP_REFERER']);
?>
```

Kuva 29. Tulosten tallennukseen tekemäni PHP-scripti on tällainen. Tuloksia kirjoitetaan kymmenen kappaletta, jolloin viimeinen tulos jää aina pois, jos uusi tulos tallennettiin.

6 TESTAUS

Pelien testaus tuli katettua melko laajasti jo ohjelmointivaiheessa eikä niistä enää löytynyt mitään vikoja varsinaisessa testausvaiheessa. Tulosten tallennusta testattaessa kävi ilmi, että tulokset eivät aina päivity sivulla. Vaikka sivun päivittäisi selaimen päivitysnapista (refresh), tulokset eivät aina silloinkaan päivittyneet. Tulosten päivityksen saa tehtyä aina varmasti kun pitää control (ctrl) nappia pohjassa päivittäessään sivun. Pikanappi sivun päivitykseen on F5, joten CTRL+F5 on nopea tapa päivittää tulokset aina ajan tasalle.

Web-sivustojen toimivuuden ainainen riesa on käyttäjien eri selaimet, niin kuin aiemmin on jo mainittu (Kappale 3). Tämän vuoksi testaus eri selaimilla on tärkeää. Sivujen sisältö ladataan Ajaxilla joten selaimen on pakko tukea sitä, jotta sivusto toimisi. Internet Explorer vaatii version 5.0 tai uudemman ja Opera version 8.0 tai uudemman Ajaxin toimintaan, Firefoxilla Ajaxin tulisi toimia kaikilla versioilla.

Testatut selaimet:

- Mozilla Firefox 3.0.1.
- Internet Explorer 6.0.X
- Opera 9.52
- Google Chrome

Testatut selaimet ovat yleisimpiä käytettyjä selaimia, sekä upouusi Google Chrome selain. Sivuston kehitys tapahtui Mozilla Firefoxia käyttäen. Suureksi mutta positiiviseksi yllätykseksi pelit toimivat kaikissa selaimissa samalla tavalla moitteettomasti. Kaiken kaikkiaan sivut näyttivät suurin piirtein samalta jokaisessa selaimessa. Odotetusti CSS-tyyleissä oli pieniä eroja, jotka olivat hyväksyttäviä. Joissakin elementeissä käytetty reunaviivan (border) tyyppi outset ei toiminut, kuin Mozillassa ja Operassa, ja niissäkin oli eroja. Sivut eivät olleet missään kahdessa selaimessa täysin samanlaiset. Seuraavilla sivuilla olevista kuvista selviää valikon ja pisteiden näkyvyydessä olevat eroavaisuudet eri selaimissa. Jokaisessa kuvassa hiiri on Muistipelilinkin päällä.

Kuva 30. Vasemmalla Mozilla Firefoxin näkymä ja oikealla Internet Explorer. Internet Explorer oli ainoa selain, jossa navigointilinkin taustaväri ei vaihtunut hiiren mennessä sen päälle.

Kuva 31. Vasemmalla Opera ja oikealla Chrome. Operassa valikko ja pisteet mahtuivat paljon matalampaan tilaan, kuin muissa selaimissa. Linkin taustaväri vaihtui oikein, mutta reunukset ovat molemmissa erilaiset. Chromessa reunukset olivat vain mustan väriset.

Navigointilinkit ovat HTML-elementtejä, joihin reunukset piirtävä CSS koodi ei toiminut kaikissa selaimissa:

```
td#left span {
...
border: 1px outset black;
...
}
```

Värin vaihtuminen hiiren tullessa linkin päälle, joka ei Internet Explorerissa toiminut:

```
td#left span:hover {
background: rgb(70,0,0);
}
```

Nämä pienet tyylierot eivät ole millään tavalla haitallisia, eikä niistä tarvitse välittää. Erot ovat lähes mitättömiä ja itse pelit kuitenkin toimivat halutulla tavalla. Käyttösuositukseksi voisi kuitenkin olla Mozilla Firefox, koska se näyttää sivut täsmälleen sellaisena, kuin alun perin on tarkoitus ollut. Testattujen selaimien vanhemmilla versioilla toimivuus voi olla kyseenalaista ja sellaisten käyttöä tulisi välttää.

Operointipaneelissa sivustoa testatessa ilmeni, että sivujen haku kestää joskus melko kauan. Muutaman kerran operointipaneeli käynnistyi itsellään kokonaan uudestaan eikä tämän syy selvinnyt. Ehkä operointipaneeli kuormittui liikaa, kun sinne siirsi jatkuvasti FTP:llä päivitettyjä versioita tiedostoista ja testasi niitä.

7 TYÖN VIIMEISTELY

Työn valmistumisen jälkeen se oli esiteltävä työn tilaajalle Timo Suvelle. Koko työ käytiin läpi ja Suvela vaikutti tyytyväiseltä lopputulokseen, mutta pieniä muutoksia tuli vielä tehtäväksi. Sivulle haluttiin bannerin lisäksi otsikkoteksti ”Tekniikkanäyttely 2008”, jonka lisäys tuli tehtyä saman tien. Suvela ehdotti myös mahdollisuutta pelata pelejä näppäimistöltä. Tätä mahdollisuutta ei tullut ajatelleeksi aiemmin työtä tehtäessä, mutta se oli erittäin hyvä idea joka tulisi lisättyä. JavaScriptissä on tapahtuma ”onkeydown”, joka tapahtuu kun jotakin nappia painetaan. Tämän tapahtuman avulla pelaamisen näppäimistöltä voi toteuttaa. Napeiksi tuli Z, X, C ja V.

Toteutus vaati uuden JavaScript-funktion, jota kutsutaan joka kerta, kun jotakin näppäimistön nappia painetaan. Tässä funktiossa painetun napin koodi luetaan ja sen jälkeen vain suoritetaan nopeus- ja muistipelien tarkistusfunktiot. Vastaan tuli kuitenkin yksi ongelma. Pelit aloitetaan painamalla lomakenappia ja tällöin nappi jää tarkennetuksi objektiksi, eikä näppäimistön painalluksia lasketa. Tämän ratkaisu vei jälleen oman aikansa. JavaScriptin elementin tarkennukseen käytettävän focus()-funktion lisäys toisiin elementteihin, jolloin tarkennuksen saisi napista pois, ei auttanut. Lopuksi ongelma ratkesi erittäin yksinkertaisella lisäyksellä. Aloitusnapin onclick-ominaisuuteen oli lisättävä pelin kutsun lisäksi funktio blur(), jolloin tarkennus napista poistuu.

```
221  onkeydown = nappi;
222
223  function nappi(e) {
224      if(window.event) {
225          keynum = e.keyCode;
226      } else if(e.which) {
227          keynum = e.which;
228      }
229      if(keynum == 90) {tarkistanopeus(1);tarkista(1);}
230      else if(keynum == 88) {tarkistanopeus(2);tarkista(2);}
231      else if(keynum == 67) {tarkistanopeus(3);tarkista(3);}
232      else if (keynum == 86) {tarkistanopeus(4);tarkista(4);}
233      else {}
234  }
```

Kuva 32. Painetut napit tallentuvat ASCII-koodina. 90=Z, 88=X, 67=C ja 86=V

Aluksi tämä ratkaisu tuntui toimivan, mutta kattavamman testauksen yhteydessä peleissä ilmeni ongelmia. Ongelmat johtuivat siitä, että molempien pelien tarkistusfunktioita kutsutaan. Napin painamistapahtumaa ei kuitenkaan voi kutsua muisti- ja nopeuspelin erillisissä sivuissa. Ainoastaan sivujen alkuperäisen pyynnön yhteydessä tulleet JavaScript-koodit toimivat, eivätkä Ajaxilla myöhemmin haetuissa sivuissa olleet JavaScriptit toimineet lainkaan. Lisäksi selaintestaus paljasti, että näppäimistön painallukset eivät toimineet Internet Explorerissa.

Näppäimistön painallukset alkoivat toimia Internet Explorerissa, kun tapahtumakutsun siirsi body-tagin sisälle. Tämän lisäksi löytyi funktio String.fromCharCode(), jolla ASCII-koodin sai muutettua suoraan vastaavaksi kirjaimeksi. Tämän lisäksi oli keksittävä keino, jolla voi kutsua vain pelatun pelin tarkistusfunktiota.

Ongelma ratkesi tekemällä asia, joka olisi pitänyt tehdä jo kauan sitten. Muisti- ja nopeuspelissä oli paljon samannimisiä muuttujia, jonka takia koko ongelma oli olemassa. Molempien pelien muuttujat oli nimettävä uudelleen niin, ettei samannimisiä muuttujia enää käytetty missään. Muistipelin muuttujiin tuli asetettua alkuliite ”muisti” ja nopeuspelin muuttujiin ”nopeus”. Tämän jälkeen pelit toimivat, vaikka molempia tarkistusfunktioita kutsuttiin näppäintä painamalla. Muistipelin käyttöliittymään tuli vielä lisättyä käyttäjän toistamien värien määrä kyseisellä tasolla, koska se oli aikaisemmin jäänyt tekemättä.

```
221  function nappi(key) {
222      if(!key) {
223          key = event.keyCode;
224      }
225      keynum = String.fromCharCode(key);
226      if(keynum == "Z") {tarkistanopeus(1);tarkista(1);}
227      else if(keynum == "X") {tarkistanopeus(2);tarkista(2);}
228      else if(keynum == "C") {tarkistanopeus(3);tarkista(3);}
229      else if (keynum == "V") {tarkistanopeus(4);tarkista(4);}
230      else {}
231  }
```

Kuva 33. Uusi versio funktiosta, joka käsittelee näppäimistön painalluksen. Funktiota kutsutaan body-tagin onkeydown-attribuutilla.

Työn loppuvaiheessa myös tietoturva-asiat tulivat puheeksi työn ohjaajan, Reino Aarisen toimesta. Työhön ei liity varsinaisesti minkäänlaista salaista tietoa, jota pitäisi suojella. Kyse oli enemmänkin siitä, että pelien tuloksia pystyy huijaamaan erittäin helposti ja se pitäisi estää. Kuka tahansa pystyy katsomaan sivustolta tulosten lähetyslomakkeen lähdekoodin sisältäen PHP-palvelimen osoitteen, johon lomake lähetetään. Näiden tietojen avulla on mahdollista tehdä oma HTML-sivu minne tahansa, joka sisältää samanlaisen lomakkeen ja pelin tulokseksi pystyy asettamaan minkä tahansa arvon. Tämä tuli todettua siten, että käyttöjärjestelmän työpöydälle luotiin HTML-tiedosto, johon lomake sijoitettiin ja tulosta muutettiin. Tämän jälkeen tallenna nappi lähetti tiedot palvelimelle, joka tallensi oikeasti täysin mahdottoman tuloksen.

Ongelman ydin on siinä, että HTML ja JavaScript-koodia ei pysty täysin piilottamaan ja ne ovat kaikkien luettavissa, jonka vuoksi ongelma osoittautui hankalaksi ratkaista. Yhtenä vaihtoehtona PHP-scriptiin olisi voinut laittaa tarkistuksen siitä, että tulos on lähetetty operointipaneelin IP-osoitteesta, jolloin ainoastaan operointipaneelin sivuilta lähetyt tulokset hyväksytään. JavaScript koodin salaamiseen löytyi myös työkalu verkosta, jolla koodi kryptattiin lukukelvottomaksi eikä lomakkeen tietoja enää voinut nähdä. Nämä ratkaisut eivät kuitenkaan estä huijaamista, koska selaimiin on olemassa kehitystyökaluja, joilla näkee kaiken koodin joka tapauksessa ja sitä voi jopa muokata suoraan sivulla. Esimerkkinä Firefoxin laajennus Firebug, jonka avulla operointipaneelista haetun sivun koodia pystyi muokkaamaan selaimessa ennen lomakkeen lähetystä. Tämän vuoksi IP-osoitteen tarkistus ja kryptauksen käyttö ei estä huijaamista.

Ratkaisumahdollisuuksia tuli harkittua monia, mutta jokainen niistä oli helposti selvitettävissä, koska ne perustuivat JavaScriptin käyttöön. Esimerkiksi heti pelin loputtua, ennen kuin tulosta on voinut huijata, todellisesta tuloksesta ja jostakin toisesta arvosta olisi voinut laskea jonkin tarkistusavaimen. Ajax-kutsua ei voi lähettää ulkoiselle palvelimelle, joten tämä tarkistusavain olisi kuitenkin lähetettävä lomakkeen mukana jolloin se on myös muokattavissa kuten muutkin lomakkeen arvot. Lopulta JavaScript-tiedosto kryptattiin jonka lisäksi IP-osoitteen tarkistusta ei tarvinnut edes toteuttaa. Se on yhdentekevää koska kehitystyökalun avulla huijaus on joka tapauksessa mahdollista. Täydellistä huijauksen estoa ei siis saavutettu.

8 YHTEENVETO

Operointipaneelin www-palvelimen ominaisuuksien rajallisuuden ja ohjelmoitavan logiikan poisjäämisen vuoksi monien asioiden toteutus oli mahdotonta. Muuta tehtävää tilalle tuli kuitenkin melko paljon, kun ottaa huomioon myös työn sisältämät teoriaan painottuvat osat, kuten esiselvityksen tekeminen. Oli hyvä että tulokset sai tallennettua, vaikka itse operointipaneelissa se ei onnistunut. Näin työhön saatiin lisää sisältöä. Jälkeenpäin voin todeta olevani melko tyytyväinen lopputulokseen. Operointipaneelin tutkimisen ja muiden asioiden selvittämisen johdosta uusia asioita tuli opittua paljon, joka oli tarkoituksena. Esimerkiksi Ajaxiin tutustuin vasta työn aikana ja innostuin myös niin sanotusta DHTML:stä (Dynamic HTML) entistä enemmän.

LÄHTEET

Peltomäki, J. & Nykänen, O. 2006. Web-selainohjelmointi. Jyväskylä. Docendo Finland Oy

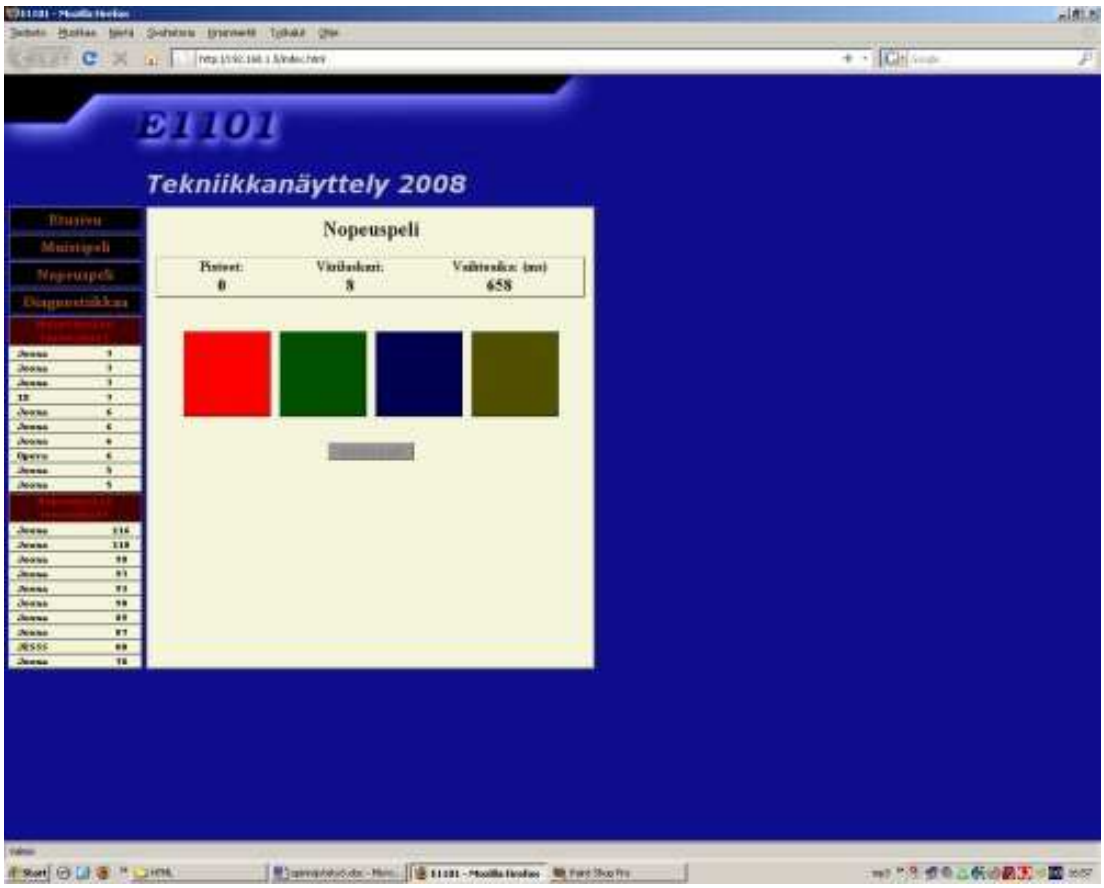
Tietoa verkkoprotokollista ja standardeista saatavilla verkosta: http://www.w3.org/

Tietoa operointipaneeleista saatavilla verkosta: http://www.beijer.fi/

E1000-sarjan operointipäätteiden englanninkielinen ohjekirja, saatavilla verkosta: http://ftc.beijer.se/files/C125728B003AF839/72984E5371134FA6C125728E005A85C7/E-Designer_for_E1000-series_English.pdf

Lähteet tarkistettu 10.10.2008

LIITE 1


E1101
Tekniikkanäyttely 2008
Nopeuspeli
Pisteet:
0
Väriä kum:
8
Vaihtuu: (ms)
658

Tekniset tiedot E1101	
Näytön koko	10,4"
Näyttötyyppi	TFT väri
Resoluutio	800 x 600 pikseliä
Näytön aktiivinen alue L x K (mm)	211,2 x 158,4 mm
Kosketustoiminto	Resistiivinen kosketusnäyttö
Toimintopainikkeet	-
Taustavalaistus	CCFL, säädettävä kirkkaus
Esitystapa	Grafiikka + teksti
Tekstin korkeus (mm)	Määriteltävissä
Valodiodeja (LED)	-
Taustavalon käyttöikä (h)	> 50 000 h
Summeri	Määriteltävä taajuus
Sovellusmuisti	12 Mb
Käyttölämpötila-alue	0 °C - +50 °C
Etupaneelin suojaus	IP66
Kotelon suojaus	IP20
EMC	EN 6100-6-2, EN 61000-6-4
UL	Kyllä
DNV	Kyllä
Syöttöjännite	+24 VDC, 20-30 V
Virrankulutus	0,5 A

Liitännät	
Sarjaportit	RS422 / RS485, naarasliitin, 25-pin D-Sub
	RS232C, urosliitin, 9-pin D-Sub
USB	Host tyyppi A (maks. 500 mA), Device tyyppi B
Ethernet	Suojattu RJ45 10/100 Mbit/s
Compact Flash	Tyypit I ja II
Mitat	
Etupaneelin mitat L x K x S	302 x 228 x 6 mm
Asennussyvyys	58 mm
Paino	2,0 kg
Toiminnot	
Kaksi laiteliityntää	Kyllä
Transparent mode	Kyllä
Passthrough mode	Kyllä
Web-toiminnot	Kyllä
Etäkäyttö Internet-selaimella	Kyllä
Monikielituki / Unicode	Kyllä / Kyllä
Reseptinkäsittely	Kyllä
Hälytyksenkäsittely	16 ryhmää
Aikakanavat	Kyllä
Reaaliaikakello	Kyllä
Historiatrendit / Tietojenkeruu	Kyllä / Kyllä
Tulostus	USB tai sarjaliitäntä
Salasanasuojaus	8 tasoa
Tuki laajennusnäppäimistölle	Näppäimistö tai hiiri USB- tai E-Key-sarjaliitännän kautta