# 2D-selainpelin suunnittelu ja ohjelmointi

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Mediatekniikan koulutusohjelma

Riihimäki, kevät 2014

Jan Kuusisto

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Riihimäki
Mediatekniikan ko
Ohjelmointi

TIIVISTELMÄ

**Tekijä** Jan Kuusisto **Vuosi** 2014

**Työn nimi** 2D-selainpelin suunnittelu ja ohjelmointi

TIIVISTELMÄ

Tämä opinnäytetyö syntyi halusta ohjelmoida ja suunnitella ohjelmia, ja kiinnostuksesta peleihin, sekä tarpeesta tuottaa käytettävyyteen ja pelattavuuteen painottunut työ videopelistä; tällaisia töitä ei tämän opinnäytetyön aloittamishetkellä löytynyt. Selainympäristö valikoitui pelin alustaksi helpon jaettavuuden, käyttöjärjestelmäriippumattomuuden ja käytettävän ohjelmointikielen, JavaScriptin, joustavuuden ja yksinkertaisuuden ansiosta.

Työn tavoitteena oli tuottaa helposti ymmärrettävä esimerkki siitä mitä pelin kehittäminen vaatii, pääpaino käytettävyydellä.

Työssä sovellettiin pelisuunnittelutietojen ja ohjelmointitietojen ja –taitojen lisäksi myös tietoja käyttöliittymien suunnittelusta, graafisesta suunnittelusta, äänen käsittelystä ja matematiikasta kaksiulotteisessa koordinaatistossa.

Työmenetelmänä oli pelin suurpiirteinen suunnittelu ja ideointi ominaisuuksista aluksi paperille, prototyypin ohjelmointi suunnitelman mukaan, uusien ideoiden ja ominaisuuksien kirjaaminen ylös myös toteutusvaiheessa ja ideoiden toteutus ja dokumentointi sitä mukaa kun uusia ominaisuuksia tai muutoksia syntyi. Muiden kirjoittamista koodiesimerkeistä oli eniten apua ohjelmointiongelmien selvittämisessä.

Tuloksena oli toimiva itse suunniteltu HTML5-peli, sekä dokumentoituja periaatteita ja muuta tietoa pelikehityksestä.

Testaaminen, erityisesti muiden suorittamana, nousi tärkeään osaan käytettävyyden ja yleisen laadun parantamisessa.

**Avainsanat** Pelit, käyttöliittymät, ohjelmointi, JavaScript, HTML5

**Sivut** 29 s. + liitteet 1 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES



Riihimäki
Degree Programme in Media Technology
Programming

**Author** Jan Kuusisto **Year** 2014

**Subject of Bachelor's thesis** Designing and programming a 2D browser game

ABSTRACT

This thesis was initiated from the desire to program and design programs, from an interest in games, as well as the need to produce a thesis focused on usability and playability; such theses were not to be found at the moment this thesis was initiated. A browser environment was selected as the platform for the game because of easy distribution, operation system independence and the flexibility and simplicity of the programming language used, JavaScript.

The objective of the thesis was to produce an easy to understand example of what it takes to develop a game, with a focus on usability.

In addition to knowledge of game design and programming, the thesis applies knowledge of user interface design, graphic design, sound processing and mathematics in a two dimensional coordinate system.

The work process consisted initially of drawing rough drafts of the game design and features on paper, programming a prototype according to the drafts, documenting new ideas and features while in the implementation phase and implementing the ideas and documenting new features or changes as they emerged. Code examples written by others were the most helpful source for resolving programming problems.

The result was a functional self-designed HTML5 game and some documented principles on developing games, and other information about game development.

Testing, especially done by others, arose as an important role in improving usability and overall quality.

**Keywords** Games, user interfaces, programming, JavaScript, HTML5

**Pages** 29 p. + appendices 1 p.

# SANASTO

| | |
|---|---|
| 2D | Two-dimensional, kaksiulotteinen |
| joystick | sauvamallinen peliohjain |
| HTML | Hypertext Markup Language, websivujen luomiseen käytettävä kuvauskieli |
| HTML5 | Viides versio HTML-kuvauskielestä, ja ensimmäinen joka tukee grafiikan piirtämistä websivuille canvakselle |
| canvas | HTML5-elementti, johon voidaan piirtää dynaamista grafiikkaa |
| JavaScript | ohjelmointikieli, jolla voidaan tuottaa dynaamista sisältöä websivulle |
| kontrollit | pelin hallintaan käytettävät näppäinyhdistelmät tai muun ohjainlaitteen liikesarjat |
| tutoriaali | havainnollistava harjoitus tai opetus |
| käyttöliittymä | user interface, käyttäjälle näkyvä osa ohjelmasta |
| törmäystarkastelu | peleissä tarkistus onko jokin kappale osunut toiseen kappaleeseen |
| sprite | 2D-grafiikassa kuva, joka esittää tyypillisesti pelihahmoja |
| fade-in | animaatio, jossa kuva muuttuu asteittain mustasta lopulliseen muotoonsa |
| pause | pelitila, jossa pelitilanne on pysäytetty, taukotila |
| game over | pelitila, jossa peli on päättynyt |
| CC0 | Creative Commons 0 –lisenssi |
| 8-bittinen ääni | chiptune eli äänipiireillä tuotettu ääni, tyypillisesti 1980-luvun videopeleissä käytetty (esim. *Super Mario Bros.*) |
| Font View | Microsoftin ohjelma, jossa voi avata fonttitiedostoja |
| heksadesimaaliluku | luku, jonka kantaluku on 16 |
| framerate | näytölle sekunnissa piirrettyjen kuvien määrä, ilmaistaan freimeinä per sekunti (FPS) |
| pikseli | yksi piste kuvassa, matkan mittayksikkö kuvaruudulla 2D-kontekstissa |

| | |
|---|---|
| freimi | yksi kuvaruutu liikkuvassa kuvassa (engl. *frame*) |
| funktio | ohjelman osa, joka toteuttaa jonkin toiminnon |
| muuttuja | tiedon tallennuspaikka ohjelmoinnissa, muuttuja voi olla esimerkiksi kokonaisluku tai merkkijono |
| boolean-muuttuja | muuttuja, jolla voi olla kaksi arvoa: true tai false, tosi tai epätosi |
| if-ehtolause | määrää toteutetaanko jokin osa koodia vai ei, jos lauseen sisällä oleva ehto toteutuu, myös siihen sidottu koodin osa toteutetaan |
| metodi | muuttujaan yhdistetty funktio |
| parametri | funktiolle välitettävä muuttuja, jonka funktio käsittelee |
| looppi | lause, joka mahdollistaa koodin osan suorittamisen useaan kertaan (engl. *loop*) |
| taulukko | array, muuttuja johon voidaan tallentaa useita muuttujia |
| jQuery | JavaScript-kirjasto |
| kirjasto | ohjelmointikielen päälle rakennettu joukko metodeja, alio-hjelmia tai luokkia |

# SISÄLLYS


1 JOHDANTO ..... 1

2 SUUNNITTELU ..... 1

2.1 Ideointi ..... 1
2.1.1 Vaikutteet ..... 2
2.2 Pelitestaus ..... 2
2.2.1 Rakenne-/prototyyppitestaus ..... 3
2.2.2 Hienosäätötestaus ..... 4
2.2.3 Hajoitustestaus ..... 4
2.2.4 Käytettävyystestaus ..... 4
2.3 Tekninen testaus ..... 4

3 PELIMEKANIIKKA ..... 4

3.1 Säännöt ..... 5
3.2 Tavoitteet ..... 5

4 KÄYTETTÄVYYS ..... 5

4.1 Kontrollit ..... 5
4.2 Ohjeistus ..... 6
4.3 Käyttöliittymätestaus ..... 8

5 ULKOASU ..... 8

5.1 Pelihahmot ..... 9
5.1.1 Pelaaja ..... 9
5.1.2 Viholliset ..... 10
5.2 Pelialue ..... 10
5.2.1 Tutoriaali ..... 10
5.2.2 Tason aloitusruutu ..... 10
5.2.3 Pause-ruutu ..... 11
5.2.4 Game Over –ruutu ..... 12
5.3 Infopalkki ..... 13
5.4 Äänet ..... 13
5.4.1 Tapahtumat ..... 14
5.5 Typografia ..... 15

6 OHJELMAKOODI ..... 15

6.1 Canvas ..... 15
6.2 Animaatio ..... 16
6.3 Pelaaja ..... 16
6.3.1 Liike ..... 17
6.3.2 Ampuminen ..... 18
6.4 Viholliset ..... 21
6.4.1 Liike ..... 21
6.4.2 Ampuminen ..... 23
6.4.3 Ominaisuudet ..... 25
6.4.4 Luonti ..... 25

6.5 Törmäystarkastelu ..... 27
7 TEKNIIKKA ..... 27
7.1 JavaScript ..... 28
8 YHTEENVETO ..... 28
LÄHTEET ..... 29

Liite 1 Enemy-luokkafunktion lähdekoodi

# 1 JOHDANTO

Reko on yhden pelaajan 2D-räiskintäpeli, jossa tavoitteena on selviytyä hengissä tuhoamalla vihollisia. Kun vaarallisimmat viholliset on tuhottu, pääsee seuraavalle tasolle. Pelissä on kymmenen tasoa, ja kun viimeinen taso on läpäisty, peli on voitettu. Pelaajalle annetaan pelin alussa kolme elämää, lisäelämiä on mahdollista saada keräämällä pisteitä. Yksi osuma viholliseen tai vihollisen ampumaan luotiin vie yhden elämän ja peli on hävitty, kun kaikki elämät on menetetty.

Opinnäytetyön tavoite oli tuottaa peli, jota voi pelata selaimella. Pelin tekemisessä on panostettu erityisesti selkeyteen ja helppokäyttöisyyteen. Tämä ei kuitenkaan tarkoita että peli olisi liian pelkistetty tai vaikeustasoltaan erityisen helppo. Pelin kontrollit on pyritty pitämään minimaalisina, jotta ne on helppo omaksua ja vaikeustaso on pyritty pitämään tarpeeksi haastavana, jotta peli pysyy mielenkiintoisena.

Pelissä ei ole varsinaista tarinaa eikä tarinankerrontaa käsitellä tässä opinnäytetyössä.

Tässä opinnäytetyöraportissa käsitellään sitä, miten pelin toiminnot on toteutettu ja mikä niiden tarkoitus on. Opinnäytetyöraportin tavoitteena on dokumentoida pelin tekemisen prosessi mahdollisimman hyvin ja antaa lukijoille ideoita ja työkaluja omien pelien kehittämiseen.

# 2 SUUNNITTELU

Projektin alusta asti tavoitteena oli tehdä pelistä yksinkertainen ja pienikokoinen, jotta se ei kaatuisi omaan monimutkaisuuteensa ja aikaa jäisi enemmän viimeistelyyn ja pelattavuuden hiomiseen. Perspektiiviksi valikoitui tästä syystä 2D.

Kun perusidea siitä mitä peli tulisi pitämään sisällään oli selvillä, aloitin testattavien osien toteutuksen aloittaen välttämättömistä osista, ensin mahdollisimman yksinkertaisesti toteutettuna.

## 2.1 Ideointi

Ideointivaiheessa otin hyvin paljon vaikutteita muista peleistä ja testasin pelejä selvittääkseni niiden hyviä ja huonoja ominaisuuksia, ja mikä tekee pelistä hyvän tai huonon. Sovelsin näitä tietoja tähän peliin.

Yhtenä hyvänä ominaisuutena voisi mainita reiluuden. Esimerkiksi jos pelin häviää tai kärsii tappioita, sen ei pitäisi koskaan johtua suunnittelu- tai ohjelmointivirheestä, vaan pelkästään pelaajan itse tekemästä virheestä. Esimerkiksi tyhjästä suoraan eteen odottamattomasti ilmestyvät viholliset eivät anna pelaajalle mahdollisuutta reagoida muutokseen ja tekevät tilanteen epäreiluksi pelaajalle.

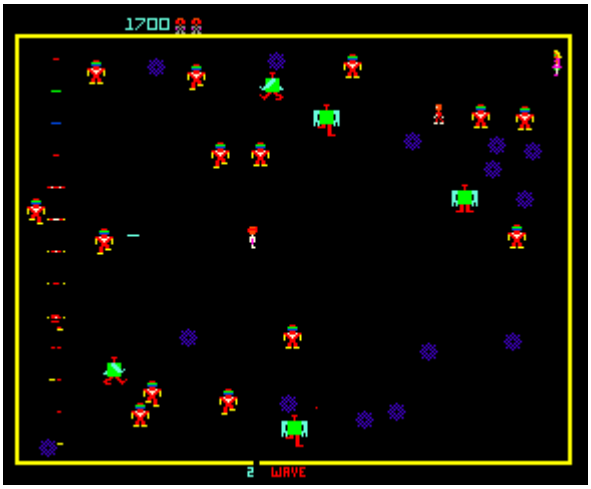
Pelaajan toiminnasta riippumattomat tilanteet, jotka pysäyttävät pelin ovat erittäin epätoivottuja. Seinän sisälle jumiutuminen on klassinen esimerkki tilanteesta jossa on mahdollista että pelaaja ei pääse eteenpäin pelissä muuten kuin käynnistämällä pelin uudestaan. Tilanne ei johdu pelaajan tekemästä virheestä, joten se on ohjelmointi- tai suunnitteluvirhe.

### 2.1.1 Vaikutteet

Vaikutteet Rekoon tulevat paljolti 1980-luvun kolikko- ja konsolipeleistä, sekä moderneista selainpeleistä.

Yksi suurimmista innoittajista tälle pelille on kolikkopeli *Robotron: 2084* vuodelta 1982. Robotronissa pelihahmoa ohjataan kahdella joystickillä, toinen liikkumista ja toinen ampumista varten. Ampuminen tapahtuu automaattisesti kun ampumissuuntaa kontrolloiva joystick osoittaa johonkin kahdeksasta mahdollisesta ampumissuunnasta. Rekossa kahden joystickin tilalla on näppäimistö ja hiiri, ja kahdeksan suunnan sijasta hiiri mahdollistaa ampumisen portaattomasti joka suuntaan, ja ampuminen tapahtuu automaattisesti kuten Robotronissakin.

Kontrollien lisäksi Rekossa on vaikutteita Robotronin säännöistä ja tavoitteista, sekä ulkoasusta. Esimerkiksi molemmissa peleissä pelimaailma on rajattu yhteen ruutuun, ja pelaaja aloittaa tason ruudun keskeltä ja viholliset sattumanvaraisesti sijoiteltuna pelaajan ympäriltä.




Kuva 1. Pelitilanne pelistä Robotron: 2084 (Vid Kidz/Williams 1982)

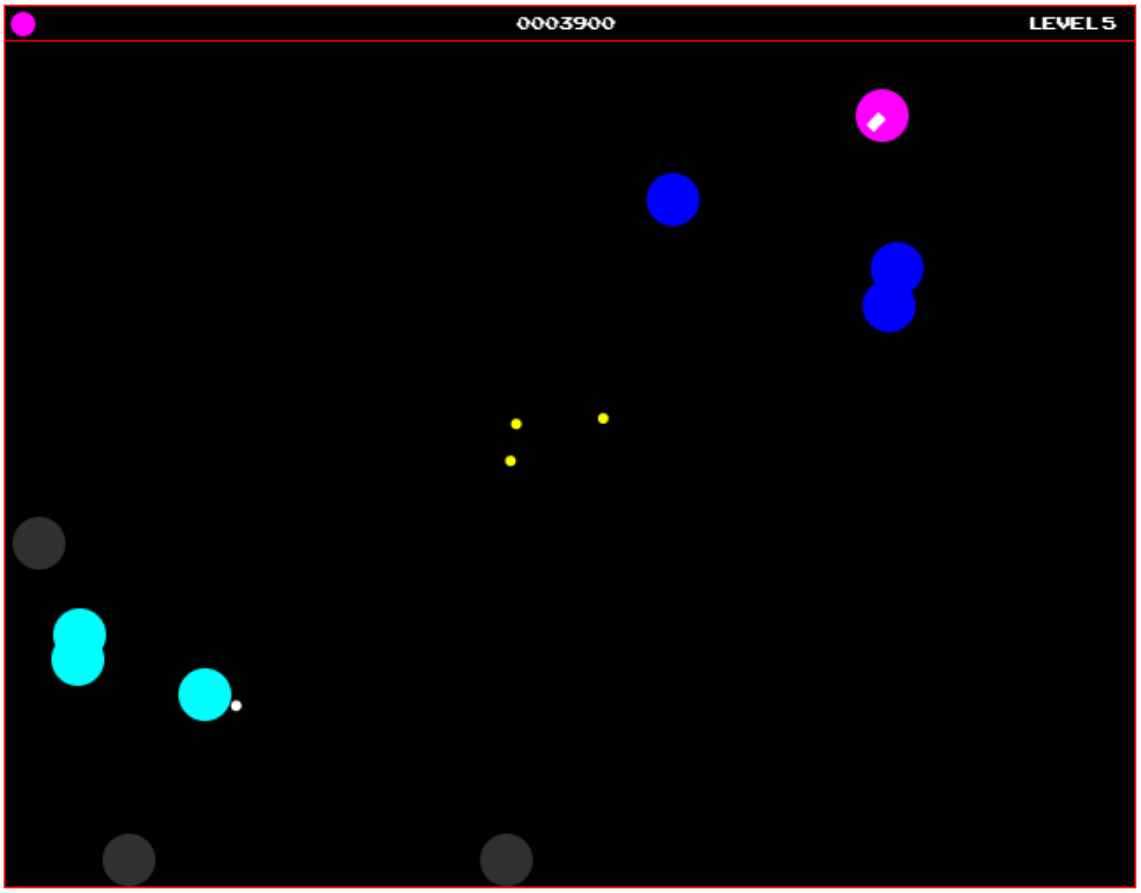
## 2.2 Pelitestaus

Suunnitteluun kuului olennaisena osana testausta, jossa kokeiltiin suunniteltujen ominaisuuksien toteutusta käytännössä. Testausvaihe myös synnytti uusia ideoita, mikäli esimerkiksi jokin paperilla suunniteltu ominaisuus oli liian monimutkainen toteuttaa. Silloin sen tilalle saattoi syntyä helpommin toteutettavissa oleva vastaava ominaisuus, mikä taas muokkaa pelin dynamiikkaa ja antaa väylän uusille ideoille.

Pelitestaus voidaan jakaa neljään eri osa-alueeseen: rakenne-/prototyyppitestaukseen, hienosäätötestaukseen, hajotustestaukseen ja käytettävyystestaukseen (Roll D6 Games 2014). Jako on peräisin lautapelien kehityksestä ja sitä voidaan soveltaa myös videopelikehityksessä. Pelien perusolemus pysyy samana ympäristöstä riippumatta, ja pelissä on tavoite tai tavoitteita, säännöt ja vastustaja tai vastustajia olipa kyseessä videopeli, lautapeli tai joukkueurheilupeli.

### 2.2.1 Rakenne-/prototyyppitestaus

Ennen kuin jokin pelin osa on lopullisessa muodossaan, on järkevää tehdä siitä ensin kevyt prototyyppi jota on helppo muokata ja jolla voi testata pelattavuutta, ennen kuin lisää siihen ominaisuuksia, yksityiskohtia tai grafiikkaa. Kuvassa 2 on pelitilanne kehitysasteelta, jossa toiminnallisuus ja esimerkiksi vihollisten käyttäytyminen on jo kehittynyt pitkälti lopullisen version tasolle, mutta pelaajan hahmoa ja vihollisia esittävät helposti piirrettävät geometriset kappaleet, tässä tapauksessa ympyrät. Valkoinen viiva pelaajan hahmossa näyttää kursorin ja ampumisen suunnan, ristikkotähtäintä ei tässä vaiheessa vielä ole.




Kuva 2. Aikaisen kehitysvaiheen pelitilanne

### 2.2.2 Hienosäätötestaus

Hienosäätötestaus pitää sisällään pelin ominaisuuksien tasapainottamista, kuten pelaajan tai muiden hahmojen liikkumisnopeudet, vihollisten määrä ja tyyppi per taso tai vaikeustason säätö. Hienosäätö ei riipu pelaajan taidoista.

### 2.2.3 Hajotustestaus

Hajotustestaus tähtää pelin rikkomiseen. Tässä vaiheessa peli olisi hyvä antaa kokeneiden pelaajien testattavaksi. Pelin rikkomisella tarkoitetaan sellaisten strategioiden tai taktiikoiden etsimistä, joilla pelin voittaa aina tai pelin kesto venyy. (Roll D6 Games 2014.)

### 2.2.4 Käytettävyystestaus

Käytettävyystestauksella selvitetään se, miten pelin oppii joku joka ei ole pelannut peliä aikaisemmin, ja miten oppimisesta voisi tehdä mahdollisimman helppoa. Esimerkiksi pelin sisäiset mahdolliset ohjeistukset voidaan luoda tällä tavalla, seuraamalla jos jokin on epäselvää testaajille ja tekemällä peliin muutoksia, joilla epäselvyydet saadaan poistettua.

## 2.3 Tekninen testaus

Videopeliympäristössä vaaditaan myös teknistä testausta kuten minkä tahansa muunkin ohjelmiston kohdalla. Käytännössä tämä tarkoittaa ohjelmointivirheiden eli bugien etsimistä, ja sen varmistamista että peli toimii odotetusti ja mahdollisesti suorituskyvyn rajojen etsintä. Pelin rajat suorituskyvyssä voidaan etsiä esimerkiksi lisäämällä vihollisten määrää niin, että peli hidastuu pelikelvottomaksi.

Myös eri selaimilla testaaminen on tärkeää HTML5-pelien tapauksessa, koska eri selaimissa on erilainen tuki HTML5:n ja JavaScriptin eri elementeille ja esimerkiksi äänitiedostoformaateille. Vaikka peli toimisikin oikein yhdessä tai kahdessa selaimessa, se ei tarkoita että se toimisi samoin myös muissa selaimissa. On mahdollista että jokin koodin osa tai ohjelmointivirhe kaataa pelin yhdessä selaimessa kun toisessa peli toimii siitä huolimatta.

# 3 PELIMEKANIIKKA

Pelimekaniikka määrittelee pelin olemuksen ilman tarinaa, sääntöjen ja tavoitteiden pohjalta. Pelin mekaniikkaan ei vaikuta esimerkiksi se, onko pelin päähahmo mies, nainen tai yleensäkään ihminen, eikä se, mihin pelin tapahtumat sijoittuvat tai mistä napista pelaaja liikkuu eteenpäin. Pelin säännöt eivät riipu siitä, miten pelihahmo tai vihollishahmot on esitetty.

Vihollisen voi esittää neliönä, natsina tai demonina, eikä se vaikuta pelin mekaniikkaan, ainoastaan tarinaan. (Lakkonen 2010.)

Esimerkiksi lasten pihaleikissä poliisi ja rosvo on tarina, joka perustelee leikin säännöt. Poliisilla on antagonisti, rosvo, jonka kiinniottaminen on leikin tarkoitus. Rosvot viedään vankilaan ja poliisit voittavat jos kaikki rosvot saadaan kiinni. Säännöt voivat vaihdella pihakunnittain, mutta on huomattavaa että itse peli ei muuttuisi vaikka poliisin, rosvon ja vankilan nimet vaihdettaisiin esimerkiksi aaveenmetsästäjäksi, aaveeksi ja eristysyksiköksi. Tarina olisi erilainen, mutta peli toimisi samalla tavalla: yksi juoksee toista takaa. (Lakkonen 2010.)

### 3.1 Säännöt

Alussa pelaajalle annetaan kolme elämää. Kun pelaaja osuu viholliseen tai vihollisen ampumaan luotiin, pelaaja menettää yhden elämän. Kun pelaaja menettää yhden elämän, taso alkaa alusta niin, että vihollisia on sen verran kuin niitä oli pelaajan kuollessa. Peli loppuu jos elämät loppuvat, ja alkaa kokonaan alusta tasosta 1. Elämiä on mahdollista saada lisää keräämällä pisteitä.

Voittotilanne on viimeisen tason läpäiseminen, tappiotilanne kaikkien elämien menettäminen.

### 3.2 Tavoitteet

Pelissä tavoitteena on selvitä seuraavalle tasolle tuhoamalla vihollisia niin pitkään kuin tasoja riittää. Toisena tavoitteena on selviytyä ja olla menettämättä elämiä, jotta eteneminen pelissä on mahdollista. Pienempänä tavoitteena on myös pisteiden kerääminen lisäelämien saamiseksi.

## 4 KÄYTETTÄVYYS

Pelistä on pyritty tekemään mahdollisimman yksinkertainen ja helppokäyttöinen, jotta kuka tahansa voi oppia pelin säännöt ja aloittaa pelaamisen vaivatta. Pelin ollessa pienikokoinen ja lyhytkestoinen oppimisen helppouden ja nopeuden tarve korostuu entisestään.

### 4.1 Kontrollit

Pelihahmoa ohjataan näppäimistöllä, joko nuolinäppäimillä tai WASD-näppäimillä. Pelihahmo voi liikkua kahdeksaan eri suuntaan, ylös, alas, oikealle, vasemmalle sekä ylä- ja alaviistoon oikealle tai vasemmalle. Ampumissuuntaa kontrolloidaan hiirellä, ja hahmo voi ampua mihin suuntaan tahansa. Ampuminen tapahtuu automaattisesti, joten hiiren näppäimiä tai muita näppäimiä ei tarvita siihen. Hiiren vasen näppäin toimii pause-näppäimenä. Pause-tilaan pääsee myös P-näppäimellä.

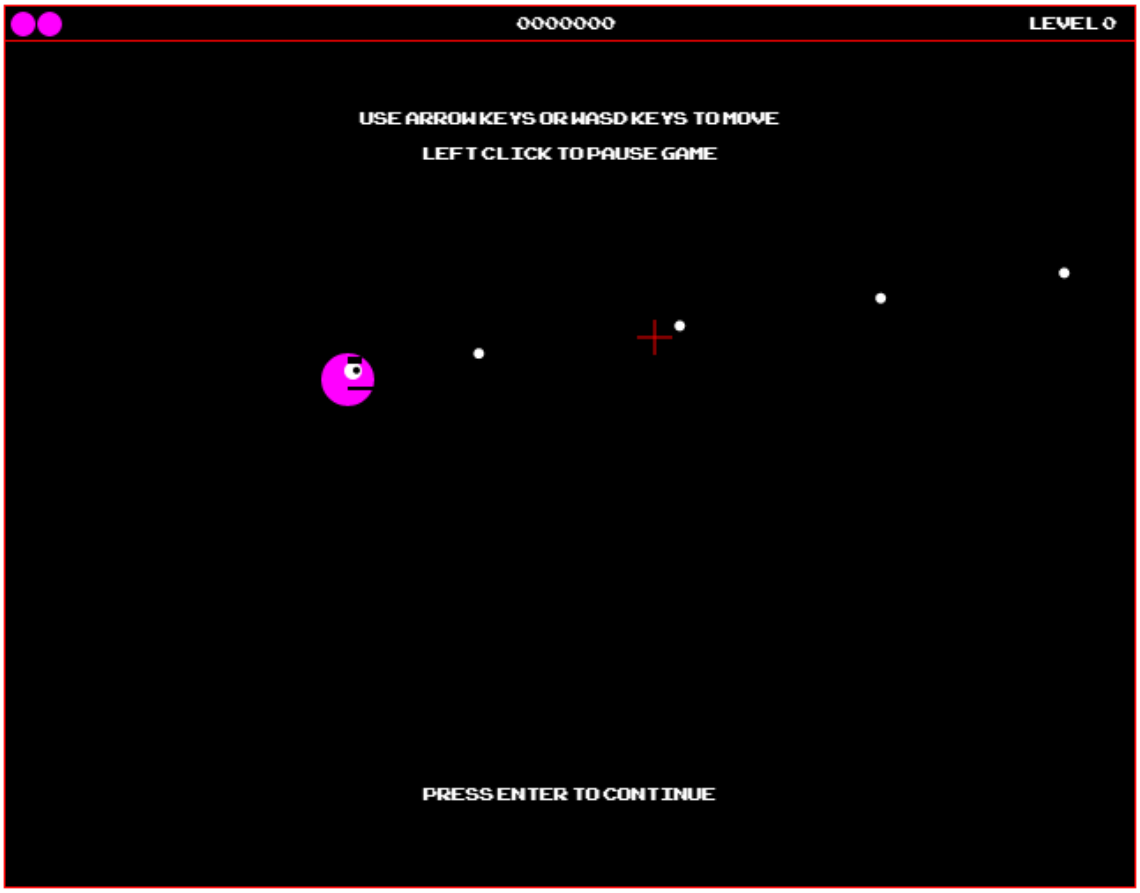
Kontrollit on tehty mahdollisimman sulaviksi, täydellinen ohjattavuus on ollut tärkeämpi prioriteetti kuin esimerkiksi pelihahmon liikkeen näyttävyys.

## 4.2 Ohjeistus

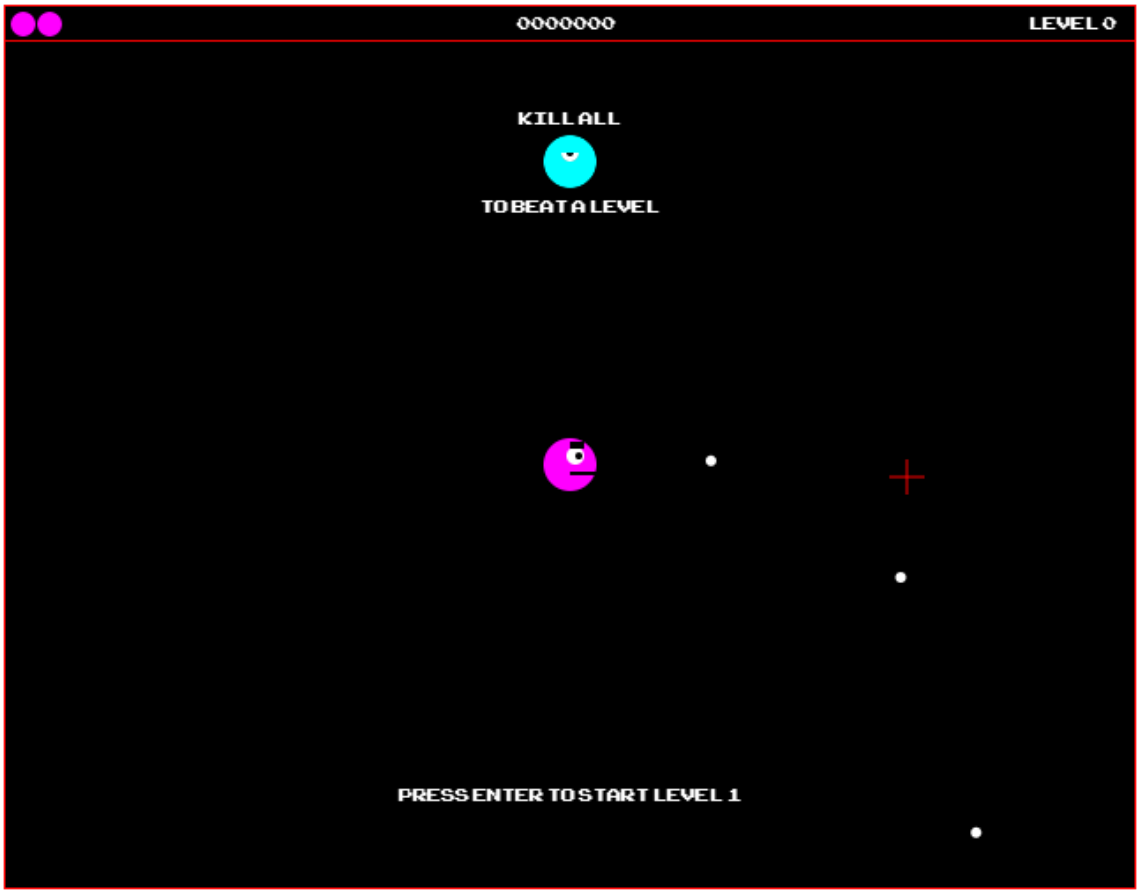
Pelin alussa on tutoriaali, jossa kerrotaan pelin kontrollit, miten hahmo liikkuu, miten ampuminen toimii, miten pelin saa tauolle ja mitkä viholliset pitää tuhota tason läpäisemiseksi.

Ohjeistus on toteutettu tekstiohjeina, kaksi riviä tekstiä kerrallaan kolmessa eri ruudussa, yhteensä siis kuusi riviä ohjeita. Samalla pelaajalla on mahdollisuus liikkua ja ampua halutessaan ja testata kontrolleja ilman vihollisia.


Kuva 3. Tutoriaalin ensimmäinen osa



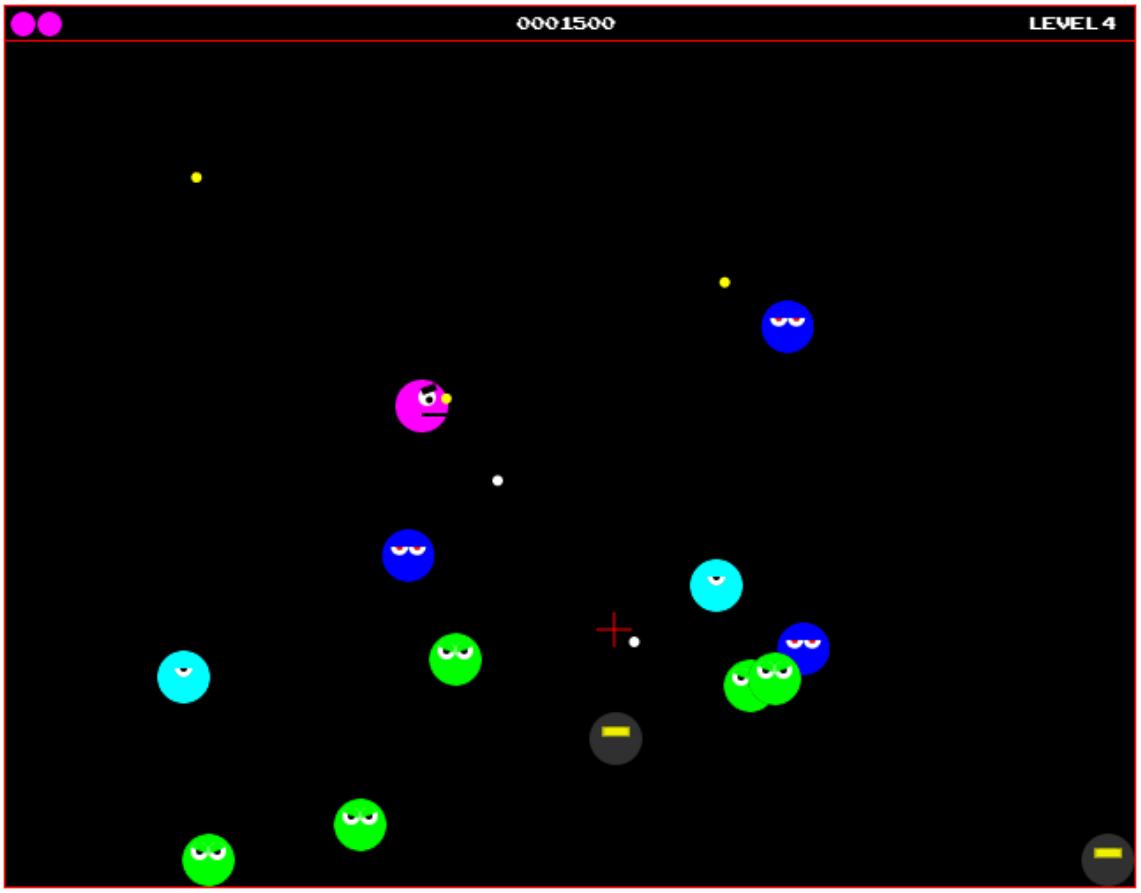

Kuva 4. Tutoriaalin toinen osa



Kuva 5. Tutoriaalin kolmas ja viimeinen osa

## 4.3 Käyttöliittymätestaus

Kun peli alkoi olla pelattava, annoin sen testattavaksi muutamalle kaverille ja tein samalla huomioita siitä, miten pelaajat reagoivat peliin ja sen ominaisuuksiin ja ovatko jotkin asiat pelissä epäselviä tai yllättäviä heille. Kirjasin testaustilanteessa ylös yllättävät ja kysymyksiä herättäneet asiat, joita voisi parantaa. Tästä syntyi esimerkiksi pelitilanteen pysäyttäminen muutamaksi sekunniksi, kun pelaajaan osuu ja hän menettää elämän, jotta tilanne ei mene ohi liian nopeasti ja että pelaaja ehtii huomaamaan mitä tapahtui. Aikaisemmassa versiossa taso alkoi välittömästi alusta kun pelaajaan oli osunut, eivätkä testaajat huomanneet heihin osuneen eivätkä tienneet mitä tapahtui.




Kuva 6. Osuma pelaajaan. Peli pysähtyy tähän näkymään muutamaksi sekunniksi ja jatkuu välittömästi sen jälkeen.

# 5 ULKOASU

Pelin grafiikka on yksinkertaista ja funktionaalista sekä helppoa piirtää ja animoida. Yksityiskohtia ei ole paljoa ja kokonaisuus pysyy selkeänä. Kaikki informaatio on kerralla näkyvissä eikä ruudun ulkopuolelta tai muualtakaan ilmesty yllättäviä uhkia.

## 5.1 Pelihahmot

Pelihahmotyyppejä on kaksi: pelaaja ja viholliset. Kaikki pelihahmot ovat samankokoisia ja samanmuotoisia, joten erottelukeinona on käytetty eri värejä ja erilaisia kasvoja kullakin hahmolla. Värit ovat mahdollisimman poikkeavia toisistaan, jotta hahmot eivät sekoitu toisiinsa nopeasti muuttuvissakaan tilanteissa. Erilaisten värien lisäksi vihollisten hahmoilla on selkeästi toisistaan erottuvat piirteet, hahmojen ollessa yksinkertaisia tämä on toteutettu kasvojen piirteitä muokkaamalla.

Pelaajan hahmo on kuvattu sivusta päin, kun vihollishahmot ovat kuvattu edestä päin, mikä saa pelaajan hahmon erottumaan paremmin muista hahmoista. Pelaajan hahmon väri myös eroaa enemmän vihollishahmojen väreistä kuin vihollishahmojen värit toisistaan paremman erottuvuuden takaamiseksi. Myös pelaajan ampumat luodit ja vihollisten ampumat luodit ovat keskenään erivärisiä erottuakseen toisistaan.

### 5.1.1 Pelaaja

Pelaaja on ympyrän muotoinen, mikä tekee törmäystarkastelusta yksinkertaisen: pelaajan osuminen seinään, viholliseen tai luotiin voidaan todeta pelihahmon eli ympyrän säteen avulla.

Pelaajan hahmolla on kolme ruumiinosaa: silmät, suu ja kulmakarvat. Nämä piirteet ovat tärkeimmät, kun halutaan ilmaista tunteita; niiden avulla voidaan esittää kaikki perusemootiot: viha, ilo, suru, pelko, inho ja hämmästys. Emootioiden käyttö tekee hahmosta inhimillisemmän ja samaistuttavamman.

Pelihahmon piirto on toteutettu kokonaan ohjelmallisesti, eli pelaajan hahmo on piirretty kokonaan ohjelmakoodilla ilman kuvia tai malleja. Pelihahmo on yksinkertainen piirtää ja se koostuu eri tilanteissa erikokoisista ympyröistä, viivoista ja puoliympyröistä. Silmä ja pupilli ovat ympyröitä ja pupilli liikkuu hiiren kursorin mukaan niin, että hahmo näyttää katsovan kursorin suuntaan, kulmakarva on viiva ja suu tilanteesta riippuen viiva tai puoliympyrä. Pelihahmo voi olla suuntautunut oikealle tai vasemmalle riippuen siitä minne suuntaan pelihahmo katsoo.

Kuva 7. Pelaajan hahmo normaalitilanteessa

Kuva 8. Pelaajan hahmo osuman jälkeen

Kuva 9. Pelaajan hahmo tason läpäisyn jälkeen

### 5.1.2 Viholliset

Kuten pelaajan hahmokin, viholliset ovat ympyrän muotoisia törmäystarkastelun yksinkertaisuuden vuoksi. Hahmoilla ei ole näkyvää kehoa, ja liikkuvien hahmojen animointi on toteutettu niin, että ne näyttävät leijuvan. Liikkeen mallintamiseksi ei tarvita erilaisia kuvia eli spriteja hahmon eri liikevaiheiden osista. Kun esimerkiksi kävelyliikkeen animoimiseksi vaadittaisiin useita spriteja, joissa hahmon jalat ovat eri asennossa, leijuvan liikkeen voi toteuttaa yhdellä spritellä, mikä helpottaa animointia.

Kuvissa 9, 10, 11 ja 12 ovat vihollisten spritet.

Kuva 10. ”Kyklooppi”, ampuva vihollinen

Kuva 11. ”Vaanija”, paikallaan pysyvä vihollinen

Kuva 12. ”Terminaattori”, tuhoutumaton vihollinen

Kuva 13. ”Zombie”, pelaajaa jahtaava vihollinen

## 5.2 Pelialue

Pelialue on rajattu ja kiinteä, se ei rullaudu pelaajan mukana, vaan koko pelialue on kerralla näkyvissä eikä se muutu. Pelaaja tai viholliset eivät voi ylittää pelialueen rajoja. Pelialue on ulkorajoja lukuun ottamatta täysin avoin, eikä siinä ole esteitä. Tausta on yksivärinen, musta.

### 5.2.1 Tutoriaali

Tutoriaali on muuten samanlainen kuin mikä tahansa taso, mutta vihollisia ei ole ja pelialueeseen on kirjoitettu ohjeet pelihahmon kontrolleihin ja pelin tavoitteisiin. Tutoriaalissa voi liikkua ja ampua täysin samanlaisesti kuin varsinaisen pelinkin aikana.

### 5.2.2 Tason aloitusruutu

Ennen kuin taso alkaa, pelitilanne on pysäytettynä muutaman sekunnin ajan, niin että ruudulla näkyy pelaaja, nykyinen taso ja viholliset jotka ilmestyvät aloituspaikoilleen. Vihollisten ilmestyminen tapahtuu fade-in-

animaationa eli ilmestyminen on animoitu niin, että viholliset muuttuvat himmeämmästä lopullisen väriseksi asteittain. Vihollisten silmät aukenevat kun fade-in on loppunut.


Kuva 14. Tason alku ja vihollisten fade-in

### 5.2.3 Pause-ruutu

Pelitilannetta ei näytetä kun peli on pysäytetty ja pause-tilassa näkyy vain kuvassa 15 näkyvä ruutu. Ruudussa kerrotaan että peli on tauolla ja miten peliin pääsee takaisin.


Kuva 15. Ruutu, joka näytetään kun peli on pause-tilassa

### 5.2.4 Game Over –ruutu

Kun pelaajan elämät loppuvat, peli on ohi ja kuvassa näkyvä ruutu näytetään pelialueella. Ruudussa kerrotaan pelin olevan ohi ja miten voi aloittaa uuden pelin.


Kuva 16. Pelin päättymisen jälkeen näytettävä ruutu

## 5.3 Infopalkki


Kuva 17. Palkki, josta selviävät pelitilannetiedot: elämät, pisteet ja taso

Infopalkista selviävät pelaajan jäljellä olevat elämät, pisteet ja tämänhetkinen taso. Se sijaitsee välittömästi pelialueen yläpuolella, erillään pelialueesta.

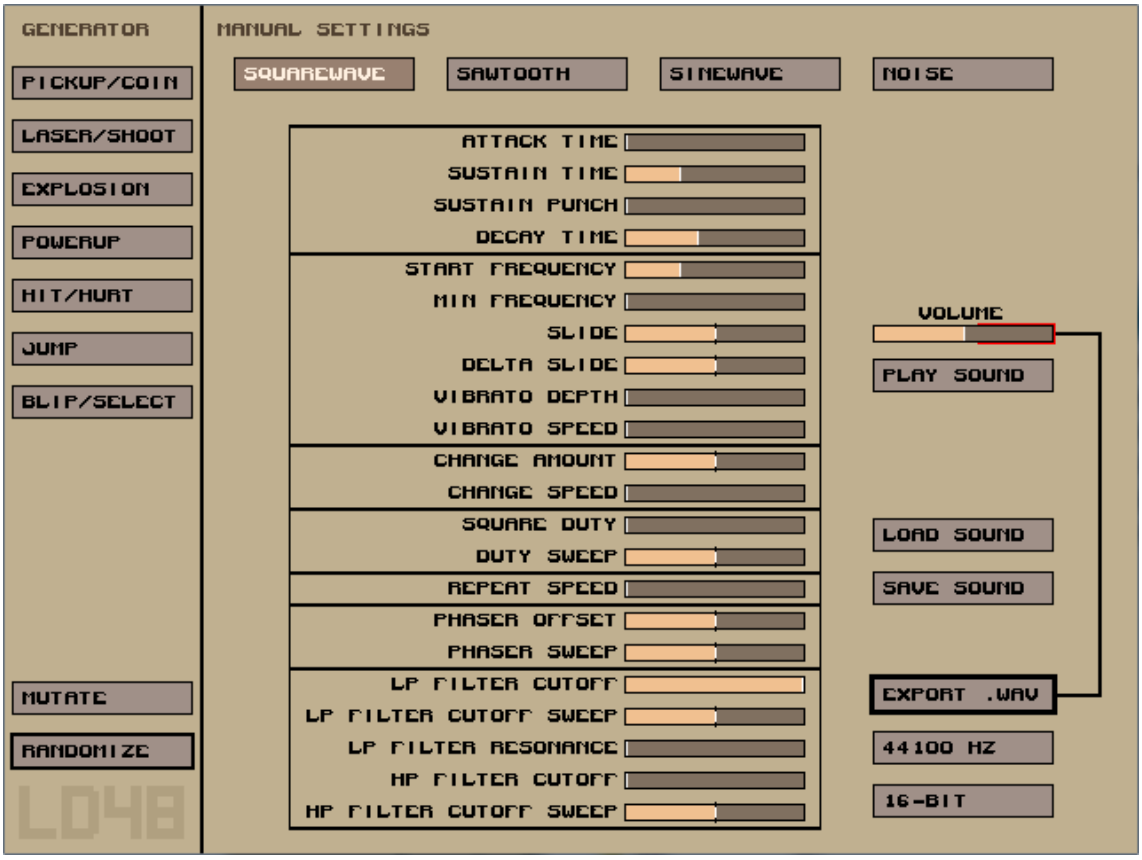
Infopalkki on sijoitettu omalle erilliselle canvakselle, sillä sitä ei tarvitse animoida kuten pelialueen canvaksen sisältöä. Infopalkin sisältö tarvitsee päivittää vain silloin, kun jokin näistä kolmesta asiasta, elämät, pisteet tai taso muuttuu. Tämä vähentää suorituskyvyn tarvetta.

Vaikka infopalkissa voi näkyä vain viisi elämää, niitä on kuitenkin mahdollista kerätä enemmän.

## 5.4 Äänet

Pelin äänitehosteet ovat lyhyitä pelitapahtumiin sidottuja ääniä, kuten ampumisen ja lisäelämän saamisen äänet. Osa äänistä on muiden tekemiä CC0-lisenssin alaisia ääniä. CC0 on Creative Commons -organisaation

määrittämä lisenssi, jonka alaiset työt ovat täysin julkisia ja kenen tahansa käytettävissä, ja työn tekijä on tätä lisenssiä käyttäessään luopunut kaikista oikeuksistaan kyseiseen työhön (Creative Commons 2014). Muut äänet ovat SFXR-ohjelmalla generoituja tai näistä generoiduista äänistä äänenkäsittelyohjelmalla muokattuja. SFXR on ilmainen ohjelma, jolla voi generoida 8-bittisiä äänitehosteita.




Kuva 18. SFXR-äänitehostegeneraattorin näkymä

Pelin äänet ovat ogg- ja aac-formaateissa. Ogg Vorbis on ilmainen avoimen lähdekoodin äänitiedostoformaatti, eikä sen käytöstä tarvitse maksaa lisenssimaksuja (Xiph.org Foundation 2008). Se ei kuitenkaan ole tuettu formaatti kaikissa selaimissa tällä hetkellä, joten kaikki äänet ovat ogg-formaatin lisäksi myös aac-formaatissa kattaakseen ne selaimet joissa Ogg Vorbis ei ole tuettu. AAC-ääniä käytetään vain, jos selain ei tue Ogg Vorbis-ääniä. AAC ei ole ilmainen äänitiedostoformaatti, mutta sen käytöstä web-sivuilla ei mene lisenssimaksuja, ja sen tuki selaimissa on sama kuin mp3-äänillä, joiden käytöstä on maksettava lisenssimaksu, joten se valikoitui toiseksi ääniformaatiksi (Via Licensing Corporation 2012).

### 5.4.1 Tapahtumat

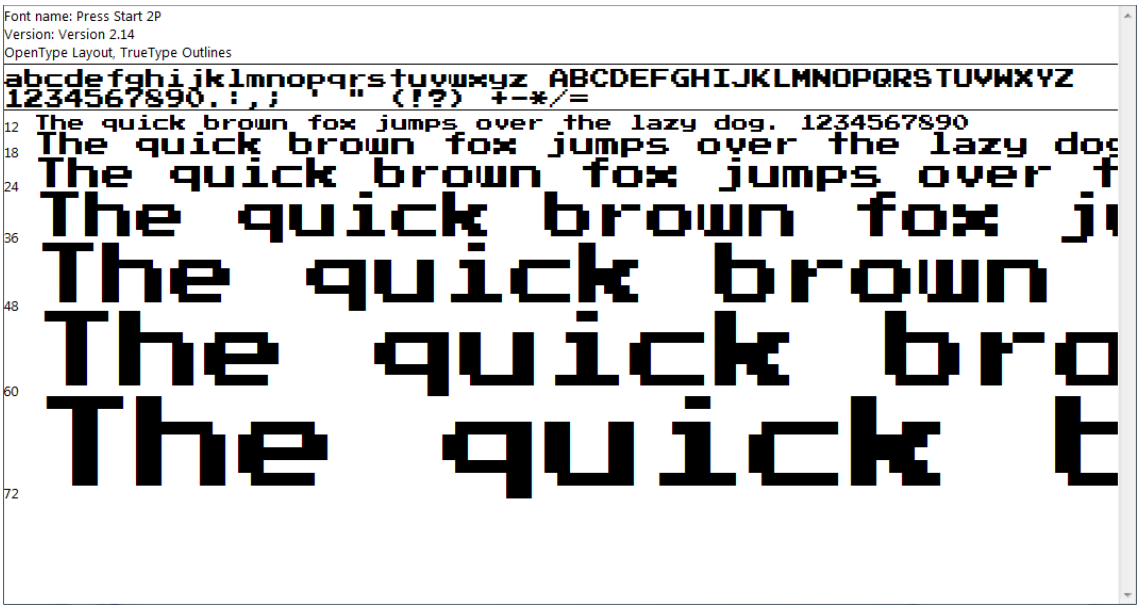
Pelissä on useita tapahtumia jotka laukaisevat äänen. Pelihahmon ampuman laukauksen, vihollisen ampuman laukauksen, pelihahmon kuolemisen, vihollisen kuolemisen ja tuhoutumattomaan viholliseen osumisen äänet ovat nauhoitettuja ääniä. Tason läpäisyn, lisäelämän saamisen ja pelin

tauolle laittamisen äänet ovat generoituja 8-bittisiä ääniä. Tason aloituksen ja game over -ruudun ääni on tason läpäisyn ääni takaperin soitettuna.

Koska kaikki viholliset ampuvat samaan aikaan, niiden ampuman laukauksen ääni soitetaan niin monta kertaa samanaikaisesti kuin ampuvia vihollisia on ampumishetkellä. Äänet soitetaan päällekkäin niin, että laukauksen ääni on sitä voimakkaampi mitä enemmän ampujia on.

## 5.5 Typografia

Pelin tekstiosat on toteutettu Press Start 2P -nimisellä fontilla, joka on avoimen lisenssin fontti ja ilmainen käyttää. Se perustuu 1980-luvun Namcon kolikkopelien fonttisuunnitteluun (Google 2014).




Kuva 19. Press Start 2P -fontti Font Viewissä

# 6 OHJELMAKOODI

Tässä luvussa käsitellään pelin oleellisimpia toimintoja ohjelmoinnin näkökulmasta. Periaatteita on havainnollistettu koodiesimerkein ja kuvin. Kaikki koodiesimerkit ovat JavaScriptillä kirjoitettuja.

## 6.1 Canvas

Pelin HTML:ssä on määritelty kaksi canvasta, toinen pelialueelle ja toinen infopalkille.

```
    <canvas id="infobar" width="640" height="20" style="border:1px solid
#FF0000; background-color:#000000; position:absolute; top:8px;"></canvas>
    <br>
    <canvas id="gamearea" width="640" height="480" style="border:1px solid
#FF0000; background-color:#000000; position:absolute; top:28px;">
    </canvas>
```

Esimerkissä canvas, jonka id on *infobar*, on infopalkin piirtoalue ja canvas, jonka id on *gamearea* on pelialue, jossa kaikki animaatio tapahtuu. Molemmissa on määritelty leveys ja korkeus pikseleinä, canvas-elementtien ulkoreunoille piirrettävä rajaviiva, taustaväri ja canvasten paikka websivulla. Värit on ilmaistu heksadesimaalilukuina, #FF0000 on punainen ja #000000 musta.

## 6.2 Animaatio

Animointi on toteutettu JavaScriptiin sisäänrakennetulla setInterval-funktiolla, joka toistaa jonkin funktion tietyin väliajoin. Framerate on määritelty alussa 50 kuvaan sekunnissa ja setInterval-funktiossa on käytetty käänteistä frameratea, joka on 1 sekunti jaettuna frameratella, jolloin kuvan päivityksen väli on 20 millisekuntia.

```
    var framerate = 50; //frames per second
    var invertFR = 1000/framerate; //animation interval in milliseconds

    function draw()
    {
      var width = canvas.width;
      var height = canvas.height;
      ctx.clearRect(0,0,width, height);
      ...
    }

    function startgame()
    {
      canvas = document.getElementById("gamearea");
      ctx = canvas.getContext("2d");
      ...
      animation = setInterval(draw, invertFR);
    }
```

Funktiossa *startgame* määritellään pelialue *canvas*, ja kontekstiksi canvakselle 2D. Canvaksen ID HTML:ssä on *gamearea*. Animaatio on muuttujafunktio *animation*, jossa suoritetaan funktio draw väliajoin *invertFR*, joka on aika millisekunteina, tässä tapauksessa 20 ms. Kaikki piirtäminen tapahtuu funktiossa *draw*, jonka ensimmäisissä riveissä määritellään pelialueen mitat, ja tyhjennetään alue, ettei edellisistä freimeistä jää jälkiä pelialueelle.

## 6.3 Pelaaja

Tässä alaluvussa käsitellään pelihahmon toiminnallisuutta. Pelihahmolla on koordinaatit, X- ja Y-piste, jotka määräävät hahmon keskipisteen. Pelihahmo piirretään ohjelmallisesti keskipisteen ympärille.

### 6.3.1 Liike

Pelaajan hahmolle on annettu nopeus, jolla se liikkuu yhtä monta pikseliä per freimi. Nopeus joko lisätään pelaajan X- tai Y-koordinaatteihin tai vähennetään niistä riippuen mihin suuntaan pelaaja on liikkumassa. Lisäys tai vähennys tapahtuu jokaisen freimin aikana jona jokin suuntanäppäin on painettuna pohjaan, joten kun pelaajan nopeus on 3 ja framerate 50 kuvaa sekunnissa, pelaaja voi liikkua maksimissaan 3*50=150 pikseliä sekunnissa yhteen suuntaan.

```
    var p1speed = 3; //player 1 speed
    function draw()
    {
    ...
      if (right) p1x += p1speed;
      else if (left) p1x -= p1speed; //if right and left keys are down, player
moves right
      if (up) p1y -= p1speed;
      else if (down) p1y += p1speed; //if up and down keys are down, player
moves up
    ...
    }
```

Koodiesimerkissä kun oikea suuntanäppäin on painettuna pohjaan, boolean-muuttuja *right* on silloin true, ja pelaajan X-koordinaattiin lisätään pelaajan nopeus, vasemman suuntanäppäimen ollessa pohjassa X-koordinaatista vähennetään pelaajan nopeus. Jos molemmat näppäimet, oikea ja vasen, ovat pohjassa, pelaaja liikkuu aina oikealle. Tämä tapahtuu siksi että oikealle ja vasemmalle liikkuminen on if ... else if –ehtojen sisällä, ja vasemmalle liikutaan vain jos *left* = true ja *right* = false. Jos taas molemmat liikkeet olisivat if-ehtojen sisällä seuraavasti:

```
      if (right) p1x += p1speed;
      if (left) p1x -= p1speed;
```

pelaaja pysyisi paikallaan kun oikea ja vasen suuntanäppäin ovat pohjassa, koska lauseet *p1x += p1speed* ja *p1x -= p1speed* kumoavat toisensa.

Pelaajan liikkuminen hetken ajan oikealle, kun molemmat vaakasuunnan näppäimet ovat pohjassa, on melko huomaamaton ominaisuus normaaleissa pelitilanteissa, mutta pelaajan pysähtymisen kokonaan huomaa välittömästi, siksi liike on toteutettu näin.

Liikkuminen ala- tai yläviistoon onnistuu, koska Y-suuntainen ylös/alas-liike ja X-suuntainen oikea/vasen-liike ovat toisistaan riippumattomia.

Seuraavassa koodiesimerkissä on näppäimistökontrollien toteutus. Jokaiselle neljästä pääsuunnasta (oikea, vasen, ylös ja alas) annetaan oma boolean-muuttuja, joka on aluksi false, ja joka asetetaan trueksi kun suuntaa vastaava näppäin on alhaalla, ja takaisin falseksi kun näppäin vapautetaan.

```
    //keyboard controls
    right = false;
    left = false;
    up = false;
    down = false;

    //set direction if corresponding keys are down
    function onKeyDown(event)
    {
      if (event.keyCode == 39 || event.keyCode == 68) right = true;
      else if (event.keyCode == 37 || event.keyCode == 65) left = true;
      else if (event.keyCode == 38 || event.keyCode == 87) up = true;
      else if (event.keyCode == 40 || event.keyCode == 83) down = true;
    }

    //and unset it when the key is released
    function onKeyUp(event)
    {
      if (event.keyCode == 39 || event.keyCode == 68) right = false;
      else if (event.keyCode == 37 || event.keyCode == 65) left = false;
      else if (event.keyCode == 38 || event.keyCode == 87) up = false;
      else if (event.keyCode == 40 || event.keyCode == 83) down = false;
    }
```

### 6.3.2 Ampuminen

Pelaaja ampuu automaattisesti tasaisin väliajoin sinne, missä hiiren kursori on. Suunta ammuksille saadaan laskettua vähentämällä pelaajan koordinaatit kursorin koordinaateista, laskemalla pelaajan ja kursorin välinen etäisyys ja normalisoimalla etäisyyden X- ja Y-suuntavektorit jakamalla ne summavektorilla.

Pelaajan ja kursorin välinen etäisyys voidaan laskea niiden X- ja Y-koordinaattien erotuksen avulla, muodostamalla suorakulmainen kolmio, jonka kateetit eli kaksi lyhyempää sivua saadaan X- ja Y-koordinaattien erotuksesta ja hypotenuusa, pisin sivu, on pelaajan ja kursorin välinen etäisyys joka saadaan laskettua Pythagoraan lauseella

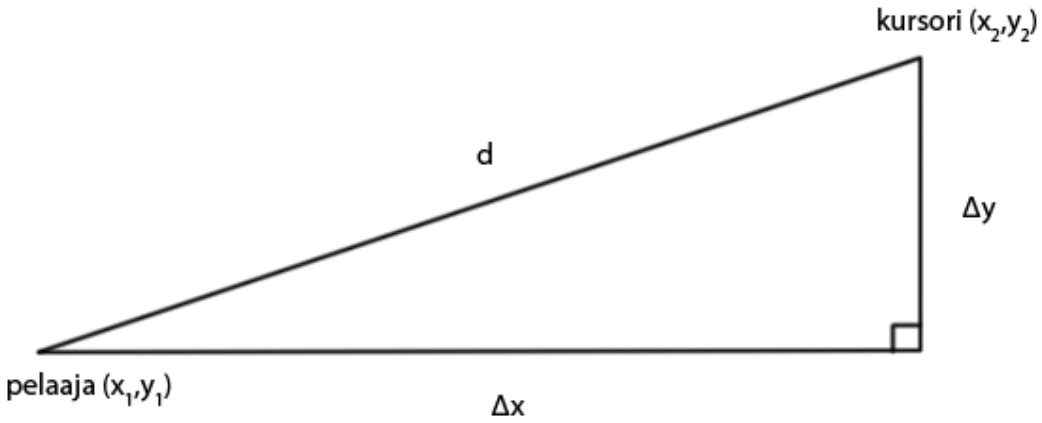
$$c^2 = a^2 + b^2, \quad (1)$$

jossa c on hypotenuusa ja a ja b ovat kateetit (Valtanen 2007, 11). Kuvassa 20 on havainnollistettu Pythagoraan lauseen käyttö etäisyyden laskemisessa, hypotenuusana on etäisyys d ja kateetteina X- ja Y-koordinaattien erotukset Δx ja Δy. X-koordinaattien erotus lasketaan kaavalla

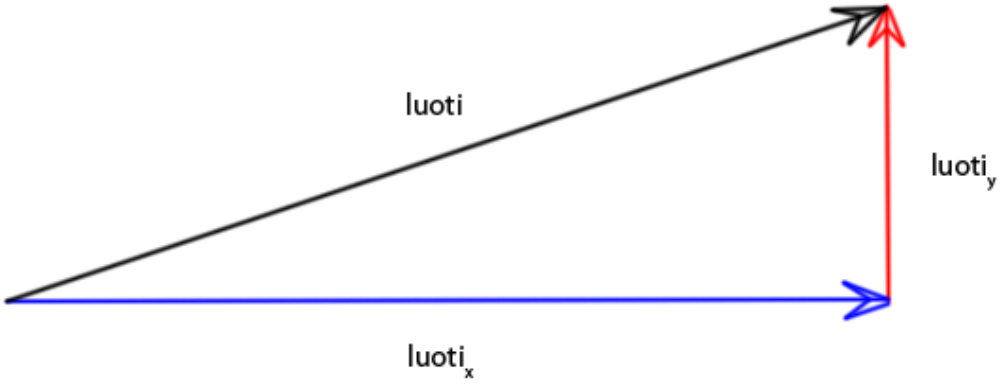
$$\Delta x = x_2 - x_1 \quad (2)$$

ja Y-koordinaattien erotus kaavalla

$$\Delta y = y_2 - y_1 \quad (3)$$




Kuva 20. Pelaajan ja kursorin välinen etäisyys d




Kuva 21. Summavektori luoti mustalla ja sen X- ja Y-suuntavektorit $luoti_x$ ja $luoti_y$ sinisellä ja punaisella

Koodiesimerkissä on oma funktio *distance(dx, dy)* etäisyyden laskemiselle, jonka parametrit *dx* ja *dy* ovat kahden kohteen väliset X- ja Y-akselin etäisyydet pikseleinä, ja joka palauttaa kohteiden välisen etäisyyden Pythagoraan lauseeseen perustuen.

```
function distance(dx, dy)
{
  return Math.sqrt(dx*dx+dy*dy);
}
```

Seuraavassa koodiesimerkissä lasketaan pelaajan ja kursorin X-koordinaattien erotus *directionX* ja Y-koordinaattien erotus *directionY*. Pelaaja sijaitsee pisteessä (*p1x,p1y*) ja kursori pisteessä (*cursorX,cursorY*). Etäisyys *lenght* saadaan *distance*-funktiolla, joka palauttaa muuttujien *directionX* ja *directionY* neliöiden summan neliöjuuren. Muuttuja *lenght* on suorakulmaisen kolmion hypotenuusa, kun *directionX* ja *directionY* ovat sen kateetit. Tämä on esitetty kuvassa 20, jossa hypotenuusa on d ja kateetit Δx ja Δy.

Pelaajan ja kursorin välisestä etäisyydestä ja näiden X- ja Y-koordinaattien erotuksista saadaan ammuttavan luodin summavektori ja X- ja Y-suuntavektorit. Kun tiedetään kuinka paljon luodin on liikuttava

X-suunnassa ja kuinka paljon Y-suunnassa, voidaan siitä laskea summavektori, joka on suorin reitti kohti kursoria (Valtanen 2007, 63) (Mäntysaari 2007). Kuvassa 21 on summavektori luoti, joka koostuu X-suuntavektorista $luoti_x$ ja Y-suuntavektorista $luoti_y$. Summavektori saadaan kaavalla

$$luoti = luoti_x + luoti_y. \quad (4)$$

X- ja Y-suuntavektorien suunta riippuu siitä, onko X- ja Y-koordinaattien erotus positiivinen vai negatiivinen luku. Jos koodiesimerkissä pelaajan X- tai Y-koordinaatti on suurempi kuin kursorin, erotus on negatiivinen. Esimerkiksi jos kursorin X-koordinaatti *cursorX* on 200 ja pelaajan X-koordinaatti *p1x* on 300, erotus *directionX* on *cursorX – p1x* = 200 – 300 = -100. Tällöin X-suuntavektori suuntautuu koordinaatistossa vasemmalle, koska vektorin lähtöpiste on pelaajan sijainti.

X- ja Y-suuntavektorit *directionX* ja *directionY* normalisoidaan jakamalla ne summavektorilla *lenght*. Normalisoitua vektoria kutsutaan myös nimellä yksikkövektori, koska sen pituus on aina 1, ja vain suunta voi muuttua. Kun suuntavektorit on normalisoitu, sillä ei ole väliä kuinka kaukana kursori on pelaajasta. Ainostaan suunnalla on merkitystä, koska normalisoitu vektori on aina samanpituinen. Tässä koodiesimerkissä normalisoitu vektori on normalisoitujen X- ja Y-vektorien *directionX/lenght* ja *directionY/lenght* summa. Vaikka X-vektorin tai Y-vektorin pituus ei ole yksinään aina 1, niiden summavektorin pituus on aina 1.

```
    function getDirection()
    {
      directionX = cursorX - p1x;
      directionY = cursorY - p1y;

      //normalization of direction
      length = distance(directionX, directionY);
      directionX = directionX/length;
      directionY = directionY/length;
    }
```

Kun suuntavektorit on normalisoitu, tiedetään mihin suuntaan luodin tulisi kulkea päästäkseen maaliin ja se voidaan ampua. Luoti lähtee pisteestä (*targetX*,*targetY*), joka on pelihahmon pupillin keskipiste. Luodilla on lähtöpisteen lisäksi X-suunta *directionX*, Y-suunta *directionY*, säde *bulletr* ja nopeus *bspeed*.

```
    function shoot()
    {
      getDirection();
      bullets.push(new Bullet(targetX, targetY, directionX, directionY, bulletr,
bspeed));
    }
```

Luodin paikka lasketaan jokaisen freimin kohdalla uudestaan, mistä syntyy luodin liike. Yhden freimin aikana luoti kulkee X-suunnassa matkan *bullet.dirX*bullet.v* ja Y-suunnassa matkan *bullet.dirY*bullet.v*. Tämä kuljettu matka lisätään luodin aiempiin koordinaatteihin. Esimerkissä *bullet.dirX* ja *bullet.dirY* ovat luodin suuntavektorit, *bullet.v* luodin nopeus ja (*bullet.x*,*bullet.y*) luodin keskipisteen koordinaatit.

```
    bullet.x = bullet.x + bullet.dirX*bullet.v;
    bullet.y = bullet.y + bullet.dirY*bullet.v;
```

## 6.4 Viholliset

Tässä alaluvussa käsitellään vihollishahmojen toiminnallisuutta, ominaisuuksia ja luontia. Vihollishahmot piirretään kuvista eli spriteista. Spritet ovat yhtä korkeita ja leveitä kuin vihollishahmojen halkaisija on, ja ne piirretään seuraavasti:

```
function drawEnemy()
{
  ctx.drawImage(enemy.sprite, enemy.x-enemy.r, enemy.y-enemy.r);
}
```

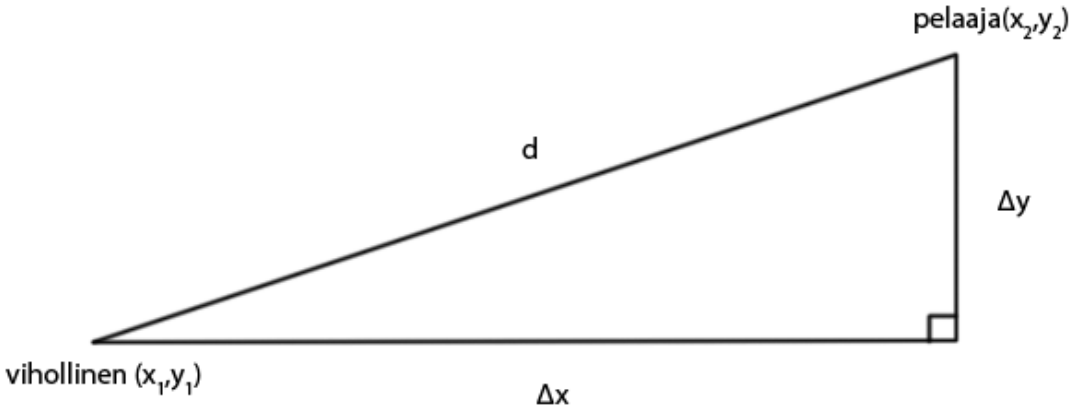
Esimerkissä *enemy.sprite* on vihollishahmoa vastaava kuva, (*enemy.x,enemy.y*) vihollisen keskipiste ja *enemy.r* vihollisen säde. Koska kuvat piirretään canvakselle vasemmalta oikealle ja ylhäältä alas alkaen kuvan vasemmasta yläkulmasta, drawImage-metodi tarvitsee kuvan vasemman yläkulman koordinaatit, jotka saadaan vähentämällä vihollisen säde *enemy.r* sen keskipisteen koordinaateista (*enemy.x,enemy.y*).

### 6.4.1 Liike

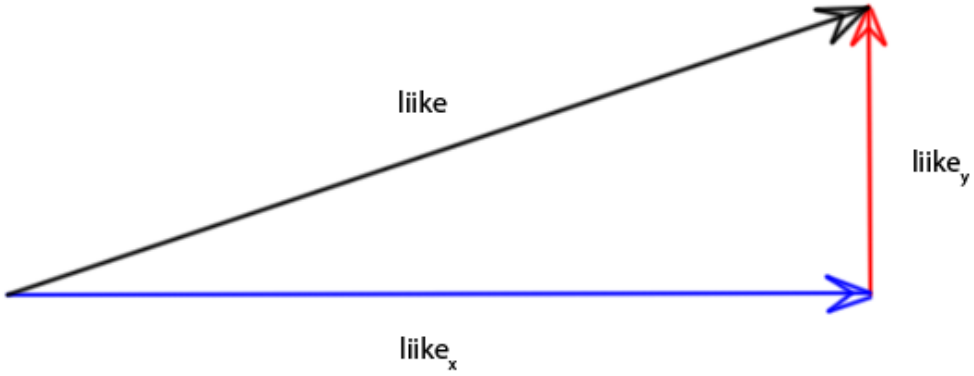
Kaikilla vihollisilla on jokin suunta ja nopeus. Jotkut viholliset pysyvät paikallaan ja näiden nopeus on nolla. Vihollisen suunta muodostuu normalisoiduista X- ja Y-suuntavektoreista. Suuntavektorit normalisoidaan jakamalla ne etäisyydellä. Vihollisen ja pelaajan välinen etäisyys voidaan laskea niiden X- ja Y-koordinaattien erotuksen avulla Pythagoraan lauseella, aivan kuten pelaajan ja kursorin välinen etäisyys aiemmassa esimerkissä.

```
function updateEnemy(enemy, enemies)
{
...
var distanceX = p1x-enemy.x;
var distanceY = p1y-enemy.y;
d = distance(distanceX, distanceY);
enemy.dirX = distanceX/d;
enemy.dirY = distanceY/d;
...
}
```

Funktiossa *updateEnemy(enemy, enemies)* lasketaan pelaajan ja vihollisen väliset X- ja Y-koordinaattien etäisyydet *distanceX* ja *distanceY*, kun pelaaja sijaitsee pisteessä (*p1x,p1y*) ja vihollinen pisteessä (*enemy.x,enemy.y*). Etäisyydet voivat olla negatiivisia, jolloin ne toimivat samalla myös X- ja Y-akselien suuntaisina suuntavektoreina. Kun tiedetään kuinka paljon vihollisen on liikuttava X-suunnassa ja kuinka paljon Y-suunnassa, voidaan siitä laskea summavektori, joka on suorin reitti kohti pelaajaa (Valtanen 2007, 63) (Mäntysaari 2007).




Kuva 22. Kahden pisteen välinen etäisyys d




Kuva 23. X- ja Y-suuntavektorit sinisellä ja punaisella sekä niiden muodostama summavektori mustalla

Toinen liiketyyppi on liikkuminen vastakkaiseen suuntaan kuin pelaaja, joka voidaan toteuttaa samalla tavalla kuin pelaajan liike, vain etumerkkejä (+ ja -) vaihtamalla. Tässä esimerkiksi kun pelaaja liikkuu vasemmalle, vihollinen liikkuu oikealle ja kun pelaaja liikkuu ylös, vihollinen liikkuu alas.

```
if (right) enemy.x -= p1speed;
else if (left) enemy.x += p1speed;
if (up) enemy.y += p1speed;
else if (down) enemy.y -= p1speed;
```

Koodiesimerkissä kun pelaaja liikkuu esimerkiksi oikealle, jolloin muuttuja *right* = true, vihollinen liikkuu vasemmalle pelaajan nopeudella *p1speed*. Eli niin kauan kun oikea nuolinäppäin/D on painettuna pohjaan, jolloin *right* = true, vihollisen X-koordinaatista vähennetään pelaajan nopeus, joka näkyy vasemmalle suuntautuvana liikkeenä.

Viholliselle voidaan lisätä prototype-ominaisuuden avulla uusi metodi *update*, jonka toiminnallisuus tulee funktiosta *updateEnemy(enemy, enemies)*, joka määrittää minne vihollinen liikkuu ja tähtää. Vihollinen on määritelty luokkafunktiossa *Enemy* (Liite 1).

```
Enemy.prototype.update = function()
{
  updateEnemy(this, enemies);
}
```

*updateEnemy*-funktiota voidaan kutsua yhteen viholliseen sidottuna kutsumalla *update*-metodia, jolloin tämä vihollinen liikkuu tai ampuu kuten *updateEnemy*-funktiossa on määritelty. Esimerkissä muuttuja *enemy* on yksi vihollinen ja sen liikkumisen mahdollistavaa metodia kutsutaan lauseella *enemy.update();*.

```
function draw()
{
   for(var i in enemies)
    {
      enemy = enemies[i];
      enemy.update();
      ...
    }
  ...
}
```

### 6.4.2 Ampuminen

Kaikki ampuvat viholliset ampuvat samanaikaisesti tasaisin väliajoin. Laukausten välinen aika pysyy samana tason alkamisen jälkeen, ja vaihtuu kun taso vaihtuu tai pelaaja kuolee. Laukausten välisen ajan vaihteluväli on 3-5 kertaa hitaampi kuin pelaajan ampumisnopeus, eli viholliset ampuvat yhden laukauksen samassa ajassa kuin pelaaja ampuu 3-5 laukausta. Tämä sattumanvarainen vaihteluväli vihollisten ampumisnopeudessa ei ole suuri, mutta tarpeeksi merkittävä, että ampumisrytmi ei ole aina sama ja liian ennalta arvattava. Tämä vaikuttaa myös vaikeustasoon niin, että sama taso voi olla vaikeampi tai helpompi riippuen vihollisten ampumisnopeudesta, mutta vaikutus ei kuitenkaan ole suuri.

Yksittäinen vihollinen ampuu kohti pelaajaa, kun pelaaja on paikoillaan, ja kun pelaaja on liikkeessä, sama vihollinen ampuu sinne minne pelaaja on menossa. Ammukset liikkuvat kuitenkin sen verran hitaasti, että pelaajalla on aikaa reagoida tähän ja harhauttaa vihollisia ampumaan harhaan. Pelaajan ollessa liikkeessä luodin maalin koordinaatit voidaan laskea kaavoilla

$$x_{pelaaja} \pm (d * v_{pelaaja})/v_{luoti} \quad (5)$$

ja

$$y_{pelaaja} \pm (d * v_{pelaaja})/v_{luoti}, \quad (6)$$

joissa $x_{pelaaja}$ ja $y_{pelaaja}$ ovat pelaajan X- ja Y-koordinaatit, $d$ pelaajan ja vihollisen välinen etäisyys, $v_{pelaaja}$ pelaajan nopeus ja $v_{luoti}$ luodin nopeus.

Viholliset eivät ammu aivan suoraan kohti pelaajaa, vaan tilaan pelaajan ympärillä, pienellä hajonnalla niin että jos pelaaja pysyy paikallaan, aivan jokainen ammus ei osu pelaajaan. Koordinaatit joihin viholliset tähtäävät ovat sattumanvaraisia, mutta kuitenkin tiettyjen rajojen sisällä pelaajan ympärillä. Hajonta on toteutettu seuraavasti:

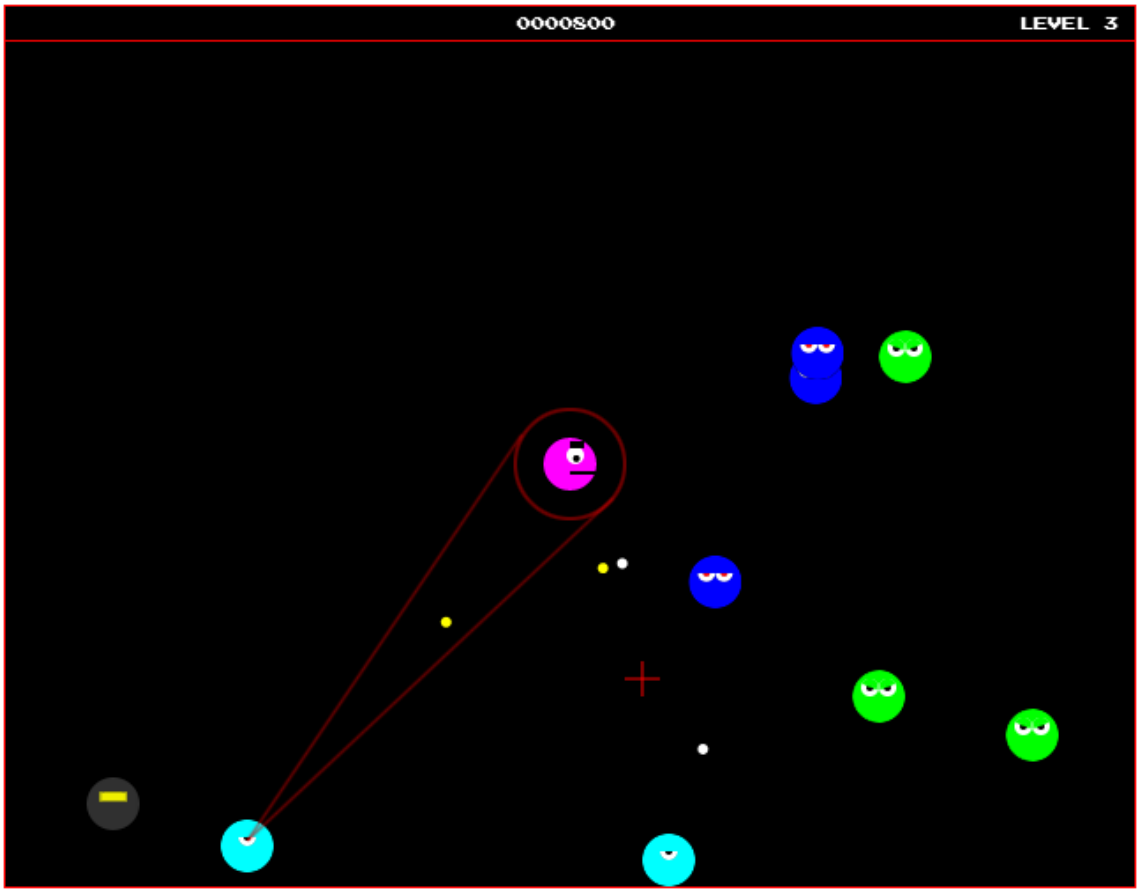
Funktiolla *randomRange(min,max)* saadaan sattumanvarainen kokonaisluku kahden arvon väliltä, parametri *min* on pienin mahdollinen satunnaisluku ja *max* suurin.

```
//get random integer between two values
function randomRange(min, max)
{
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

Seuraavassa esimerkissä alueen, johon viholliset ampuvat, säde on *randomTargetR*, joka on kaksi kertaa pelaajan hahmon säde *p1r*. Vihollisten tähtäämisestä vastaavassa funktiossa *updateEnemy* jokaisen ampuvan vihollisen jokaiselle ampumakerralle lasketaan eri satunnaisarvo *targetRange*, jonka pienin arvo on tähtäysalueen säde *randomTargetR* kertaa miinus yksi eli säteen vastaluku, negatiivinen arvo ja suurin arvo tähtäysalueen säde *randomTargetR*. Jos pelaajan säde on 15, niin *randomTargetR* saa arvon 15*2 = 30, ja *targetRange* voi olla mikä tahansa kokonaisluku väliltä -30 ja 30. Silloin pisteen (*randomTargetX*,*randomTargetY*), johon viholliset tähtäävät, X-koordinaatti on vähintään pelaajan X-koordinaatti *p1x* – 30 ja korkeintaan *p1x* + 30 ja Y-koordinaatti välillä pelaajan Y-koordinaatti *p1y* – 30 ja *p1y* + 30.

```
var randomTargetR = p1r*2; //range around player to which enemies shoot to

function updateEnemy(enemy, enemies)
{
  ...
  var targetRange = randomRange(randomTargetR*(-1), randomTargetR);
  var randomTargetX = p1x+targetRange;
  var randomTargetY = p1y+targetRange;
  ...
}
```




Kuva 24. Hajonta ampumisessa. Viholliset ampuvat kohti aluetta, joka on havainnollistettu punaisena kehänä pelaajan ympärillä

### 6.4.3 Ominaisuudet

Erilaisilla vihollisilla on erilaisia ominaisuuksia. Yksi vihollistyyppi, kyklooppi, osaa ampua ja terminaattoria ei voi tuhota. Erityyppisistä vihollisista saa eri määrän pisteitä. Pisteillä on merkitystä pelissä etenemisessä, sillä pisteitä keräämällä voi saada lisäelämiä. Ominaisuudet on määritelty luokkafunktiossa *Enemy(x, y, r, type)* (Liite 1).

### 6.4.4 Luonti

Vihollisten luomiselle on oma funktionsa, jonka parametrit ovat vihollisen tyyppi ja lukumäärä.

```
    function createEnemies(amount, type)
    {
      for(var i = 0; i < amount; i++)
      {
        getStartingCoordinates();
        enemies.push(new Enemy(enemyStartX, enemyStartY, enemr, type)); //spawn
enemies
        enemy = enemies[i];
      }
    }
```

Funktio luo vihollisen satunnaiseen pisteeseen *(enemyStartX,enemyStartY)* pelialueella niin monta kertaa kuin lukumääräksi *amount* on määritelty. Viholliset ja niiden ominaisuudet tallennetaan taulukkoon *enemies*. Lisäksi viholliselle annetaan säde *enemr* ja tyyppi *type*, joka määritellään *createEnemies*-funktiota kutsuttaessa, kuten funktiossa *startLevel(lvl)*.

```
    function startLevel(lvl)
    {
      switch(lvl)
      {
            ...
        case 2:
          createEnemies(3, "zombie");
          createEnemies(3, "lurker");
          createEnemies(2, "cyclops");
          break;
           ...
      }
    }
```

Esimerkkinä tason 2 alussa luodaan 3 vihollista joiden tyyppi on zombie, 3 vihollista joiden tyyppi on vaanija ja 2 vihollista joiden tyyppi on kyklooppi.

Tason aloitusfunktion lisäksi *createEnemies*-funktiota käytetään myös, kun taso aloitetaan uudestaan pelaajan menettäessä elämän.

```
    function restartLevel()
    {
      ...
      var remainingEnemy = []; //types of remaining enemies are saved into this
array before the enemies are erased
      for(var i = 0; i < enemies.length; i++)
      {
        enemy = enemies[i];
        remainingEnemy[i] = enemy.type;
      }
      enemies = []; //erases any enemies currently on screen
      for(var i = 0; i < remainingEnemy.length; i++)
      {
        createEnemies(1, remainingEnemy[i]); //every enemy that was left will be
recreated one by one
      }
    }
```

Funktiossa luodaan taulukko *remainingEnemy*, johon tallennetaan jäljellä olevien vihollisten tyyppi. Jäljellä olevien vihollisten lukumäärä saadaan *enemies*-taulukon pituudesta, ja väliaikaistaulukosta *remainingEnemy* tulee yhtä pitkä kuin vihollisia on jäljellä. Taulukko *enemies* tyhjennetään ja viholliset luodaan uudestaan *createEnemies*-funktion ja for-loopin avulla niin. että viholliset luodaan yksi kerrallaan ottamalla niiden tyyppi *remainingEnemy*-taulukosta, johon ne tallennettiin ennen tason aloittamista alusta. Taso alkaa uudestaan siitä tilanteesta mihin se jäi, samoja vihollisia ei tarvitse tuhota uudestaan.

## 6.5 Törmäystarkastelu

Törmäystarkastelu peleissä on yksinkertaisesti tarkistus onko jokin kappale osunut toiseen kappaleeseen, esimerkiksi onko pelaaja osunut viholliseen tai seinään.

```
    function draw()
    {
    ...
        //enemy-bullet hit detection
        for(var j in bullets)
        {
          for(var i in enemies)
          {
            bullet = bullets[j];
            enemy = enemies[i];
            diffX = bullet.x - enemy.x;
            diffY = bullet.y - enemy.y;

            if(distance(diffX,diffY)<=bullet.r+enemy.r)
            {
              if(enemy.destroyable==true)
              {
                removeEnemy(enemies, enemy);
                removeBullet(bullets, bullet);
    ...
              }
              else removeBullet(bullets, bullet);
            }
          }
        }
    ...
    }
```

Esimerkissä tarkistetaan onko pelaajan ampuma luoti osunut viholliseen. Koska sekä viholliset että luodit ovat ympyrän muotoisia, törmäystarkastelu voidaan tehdä käyttäen vihollisten ja luotien keskipisteitä ja säteitä. Ensin lasketaan luodin ja vihollisen keskipisteiden X- ja Y-koordinaattien väliset etäisyydet *diffX* ja *diffY*. Näistä saadaan Pythagoraan lauseella luodin ja vihollisen välinen etäisyys *distance(diffX,diffY)*. Jos tämä keskipisteiden välinen etäisyys on pienempi kuin luodin ja vihollisen yhteenlasketut säteet *bullet.r+enemy.r*, luoti on osunut viholliseen.

Jos luoti osuu viholliseen, tarkistetaan onko vihollinen tuhottava lauseella *if(enemy.destroyable==true)*. Jos on, poistetaan sekä osunut luoti että vihollinen johon osui. Muussa tapauksessa poistetaan pelkästään osunut luoti.

# 7 TEKNIIKKA

Peli on kirjoitettu JavaScriptillä hyödyntäen HTML5:n canvas-elementtiä, johon voidaan piirtää dynaamista grafiikkaa web-sivulla, ilman selaimen lisäosien tarvetta. Peli toimii kaikilla HTML5:n canvas-elementtiä tukevilla selaimilla, joita ovat Chrome, Firefox, Internet Explorer 9 ja sitä uudemmat versiot, Safari ja Opera. Pelin toiminta vaatii myös näppäimistön ja hiiren tai vastaavan laitteen.

Kehitystyökaluna on käytetty ilmaista PSPad-tekstieditoria.

### 7.1 JavaScript

JavaScript on web-ympäristössä käytettävä ohjelmointikieli, jolla voidaan tuottaa dynaamista sisältöä web-sivuille.

Peli on kirjoitettu lähes kokonaan puhtaalla JavaScriptillä ilman kirjastoja. Ainoastaan näppäinkontrollien sitominen on toteutettu käyttäen jQueryä, joka on JavaScriptin päälle rakennettu yleiskirjasto.

## 8 YHTEENVETO

Opinnäytetyön tavoitteena oli tuottaa valmis ja pelattava HTML5-selainpeli, mikä onnistui. Toinen tavoite, pelin tekemisen prosessin dokumentoiminen helposti ymmärrettävään muotoon osoittautui haastavammaksi. Kokonaisen pelin suunnittelu ja toteutus on laaja aihe ja työssä tehtyjen valintojen selittäminen ja perustelu vaativaa.

Työn edetessä ilmeni, että testaaminen on tärkeä osa sovelluskehitystä ja sama pätee myös peleihin. Vain testaamalla riittävästi voidaan löytää ja korjata ongelmakohdat ja tuottaa laadukas ja käytettävyydeltään mukava peli. Aivan erityisesti painotan sitä, että peliä testaavat jo kehitysvaiheessa muutkin kuin pelin kehittäjät, koska pelin kehittäjä on liian tottunut omaan tuotokseensa ja osaa odottaa pelin tekemiä odottamattomiakin asioita, jotka tulevat muille yllätyksenä. Tähän työhön sain paljon hyviä ideoita ja parannusmahdollisuuksia kun muut testasivat peliä. Mitä useampia ja erilaisempia testaajia pelillä on, sitä todennäköisemmin saadaan kerättyä tietoa pelin ongelmakohdista ja korjattua ne.

Suurimpina haasteina työssä olivat grafiikan piirtäminen ja valmiiden metodien vähäisyys JavaScriptissä. Grafiikan piirtämistä JavaScriptissä hankaloitti muun muassa X- ja Y-koordinaattien käsittely erillisinä arvoina pisteiden sijaan, ja vektorien puuttuminen. Vaikka piirto-ominaisuudet JavaScriptissä ovat monipuoliset ja joustavat, monia muista ohjelmointikielistä löytyviä yksityiskohtaisempia metodeja puuttui, mikä on ymmärrettävää skriptikielen tapauksessa. Grafiikan piirtämiseen on olemassa ulkopuolisia kirjastoja, joita tässä työssä ei käytetty.

Pelin graafista ilmettä voisi vielä parantaa ja hahmojen ulkoasua monipuolistaa, sekä erityisesti lisätä animaatiota hahmojen liikkeeseen ja lisätä hahmoille jalat tai muita liikkuvia osia. Peliin voisi lisätä myös variaatiota kiinnostavuuden lisäämiseksi. Pelin kehitys jatkunee tämän opinnäytetyön jälkeenkin.

# LÄHTEET

Creative Commons. 2014. Creative Commons – CC0 1.0 Universal. Viitattu 23.5.2014. https://creativecommons.org/publicdomain/zero/1.0/

Google. 2014. Google Fonts Press Start 2P. Viitattu 16.4.2014. https://www.google.com/fonts/specimen/Press+Start+2P

Lakkonen, T. 2010. Pelimekaniikka. Viitattu 8.5.2014. https://wiki.metropolia.fi/display/~m0502452/Pelimekaniikka

Mäntysaari, H. 2007. Ohjelmoijan matematiikka: Osa 3 – Vektorit. Viitattu 12.3.2014. http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=mat3

Roll D6 Games. 2014. Pelitestaus. Viitattu 1.4.2014. http://www.rolld6.com/pelikehittajalle/pelitestaus/

Valtanen, E. 2007. Matematiikan ja fysiikan käsikirja. Jyväskylä: Gummerus

Via Licensing Corporation. 2012. AAC Frequently Asked Questions. Viitattu 26.5.2014 http://www.vialicensing.com/licensing/aac-faq.aspx

Xiph.org Foundation. 2008. Vorbis.com: FAQ. Viitattu 26.5.2014. http://www.vorbis.com/faq/#sell

Liite 1

ENEMY-LUOKKAFUNKTION LÄHDEKOODI

```
function Enemy(x, y, r, type)
    {
      this.x = x;
      this.y = y;
      this.r = r;
      this.type = type;
      this.dirX = 0;
      this.dirY = 0;
      this.v = enemyspeed;
      this.canShoot = false;
      this.destroyable = true;

      if(this.type=="zombie")
      {
        this.color = "#0000FF";
        this.points = 100;
        this.sprite = zombieSprite;
      }
      if(this.type=="lurker")
      {
        this.color = "#00FF00";
        this.points = 50;
        this.sprite = lurkerSprite;
      }
      if(this.type=="cyclops")
      {
        this.color = "#00FFFF"
        this.points = 200;
        this.canShoot = true;
        this.sprite = cyclopsSprite;
      }
      if(this.type=="terminator")
      {
        this.color = "#303030"
        this.points = 0;
        this.destroyable = false;
        this.sprite = terminatorSprite;
      }
    }
```