Peliprojektin toteutus Phaserilla

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

HAMK Riihimäki Tieto- ja viestintätekniikka

Kevät 2019

Mikko Mäki-Kerttula

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
HAMK Riihimäki

Tekijä Mikko Mäki-Kerttula **Vuosi** 2019

Työn nimi Peliprojektin toteutus Phaserilla

Työn ohjaaja/t Petri Kuittinen

TIIVISTELMÄ

Pelit ovat kehittyneet huimasti niiden alkuajoista ja käyn läpi pelejä eri vuosilta. Valitsin pelit siten, että pääsemme lähemmäs pelejä, mistä itse olen ottanut inspiraatiota omaan peliprojektiini.

Tavoitteena oli tuottaa toimiva peli käyttäen jotain valmista pelimoottoria.

Kävin yleisesti läpi erilasia pelinkehitysympäristöjä kuten Unity, Unreal Engine, Phaser ja Godot. Toin esille niiden hyvä ja huonoja puolia ja miksi itse päädyin käyttämään Phaseria.

Itse peliprojektissa kerron, miten ohjelmointini eteni ja mitä pelin osia sain milloinkin tehtyä. Tuon samalla esiin monia ongelmia, mitä kohtasin ohjelmoinnissa.

Lopuksi käyn läpi koko projektin ja miten sen eri osa-alueet onnistuivat ja mitä puutteita siihen jäi.

Avainsanat JavaScript, peliohjelmointi, Bullet Hell

Sivut 54

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and communication technologies
HAMK Riihimäki

Author Mikko Mäki-Kerttula **Year** 2019

Subject Game project with Phaser

Supervisors Petri Kuittinen

ABSTRACT

Games have evolved greatly from their early days and I'm going to highlight games from different years. I chose games that are going towards the style of game I created in this project.

The aim was to create a working game using a game engine.

I go through different game programming systems like Unity, Unreal Engine, Phaser and Godot. Highlighting some of their positive and negative aspects and why I chose the engine I did.

In the game project I tell how my programming proceeded and what parts I did at what time. At the same time, I bring forward problems I encountered.

Lastly, I go through the whole project and what I succeeded and what shortcomings I encountered.

Keywords JavaScript, game programming, Bullet Hell

Pages 54

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on toteuttaa 2D-peli käyttäen Phaser-pelimoottoria. Aihe on valittu oman mielenkiinnon kautta. Olen ollut kiinnostunut pelien tekemisestä ja oman pelin toteuttaminen on ollut yksi tavoitteistani. Päädyin 2D-peli ympäristöön, koska olen aloittelija pelien kehittämisessä.

Päädyin toteuttamaan Bullet Hell -tyylisen shoot 'em up -pelin, koska se vaikutti genreltä, missä pystyisin samalla tekemään valmiin pelin ja haastamaan itseni kunnolla. Bullet Hell -peleissä tärkeintä on, että ruudulla on mahdollisimman monta ammusta, mitkä voivat tappaa pelaajan. Tästä syystä suuri osa pelin toteutusta kului erilaisten ammusratojen suunnittelussa, jotta ne olisivat mahdollisimman vaikea väistää, mutta ei silti niin vaikea, että pelaaminen menisi liikaa tuurin puolelle.

1.1 Pelin tavoitteet

Pelien kehittäminen on muuttunut helpommaksi päästä alkuun, koska monet pelimoottorit, kuten Unity ovat helpottaneet pelin ohjelmointia niin paljon, että asiaan tutustumatonkin voi oppia pelin teon pääpiirteet helposti. Myös videopalveluiden yleistyminen informaation välitys välineenä on helpottanut pelimoottorien alkeiden opettelemista. Ensimmäiseksi peliprojektikseen voi seurata jonkun muun tekemän video tutoriaalin alusta loppuun. Tämä kuitenkin tarkoittaa, että pelimoottoreista on vaihtelevasti sisältöä tarjolla. Suosittuihin pelimoottoreihin löytyy paljon sisältöä ja ei niin suosittuihin löytyy vähemmän. Tämä vaikutti omaan pelimoottorin valintaani. Phaserista löytyy hyvin sisältöä niin dokumentaation kautta ja vain ihan googlaamalla sen hetkisen ongelmansa. Phaser on myös JavaScript pohjainen pelimoottori, mikä tarkoitti, että minun ei tarvinnut opetella uutta ohjelmointi kieltä samalla kun keskityin pelin tekemiseen. Olimme myös käyttäneet Phaseria jo koulussa opinnoissa, joten olin valmiiksi omaistanut sen pääpiirteitä.

Opinnäytetyön toisessa kappaleessa keskityn yleisesti pelien historiaan. Aloitamme peleihin lähestymisen mahdollisimman kaukaa ja lähdemme siitä haarautumaan kohti omaa peliaihe genreäni. Miten pelit ovat muuttuneet vuosien aikana, miten eri peli genret ovat saaneet alkunsa ja annan peli esimerkkejä, jotka ovat asiallisia mihinkin aikaan.

Opinnäytetyön kolmannessa kappaleessa tutustumme erilaisiin pelimoottoreihin ja listaamme niiden ominaisuuksia ja miksi päädyin toteuttamaan pelini käyttäen Phaseria.

Opinnäytetyön neljännessä kappaleessa käymme läpi itse pelin toteutuksen. Aloittaen itse peliaiheen rajaamisesta itse pelin toteutukseen.

Alustavat pelin tavoitteet.

- 2D
- Bullet Hell
- 2 Bossia
- 10 tai enemmän vihollistyyppiä
- Hienoja ammusratoja
- Mahdollisimman paljon ammuksia ruudulla
- Touhou pelisarja henkisenä mallina
- 2 tasoa

Opinnäytetyön viidennessä kappaleessa pohdin, miten pelin kehitys meni. Missä asioissa onnistuttiin ja missä olisi voitu toimia paremmin. Tuliko vastaan esteitä, mitä ei pystytty ylittämään ja mistä tämä johtuu.

2 PELIEN HISTORIA

2.1 Mikä on videopeli

Cambridgen sanakirja antaa videopelille kaksi määritystä: “a game in which the player controls moving pictures on a screen by pressing buttons” ja “a game in which the player controls moving pictures on a television screen by pressing buttons or moving a short handle.” (Cambridge Dictionary, n.d) Näistä määritelmistä voidaan päätellä, että videopeleille ominaista on itse pelaajan vaikutus valta pelin kulkuun. Samalla videopelien oletetaan olevan digitaalisia ja jonkinlaisella näyttölaitteella pelattavia pelejä.

Kuva 1. Videopeli (Wikipedia 2016)

2.2 **Ensimmäinen videopeli**

Kun puhutaan ensimmäisestä videopelistä, on erimielisyyttä, mikä niistä lasketaan ensimmäiseksi. Historiallisesti ensimmäisenä videopelinä pidetään Thomas T. Goldsmith Jr. ja Estle Ray Mann kehittämää Cathode-Ray Tube Amusement Device (Katodisädeputki-viihdelaite) (1947). Peli sai inspiraationsa toisen maailmansodan tutkanäytöstä. Pelin päämääränä oli ampua ohjuksia kohteisiin. Pelin suuren tuotantohinnan takia sitä ei ikinä julkaistu myyntiin.

Cathode-Ray Tube Amusement Device on ensimmäinen patentoitu sähköinen peli ja sitä pelattiin oskilloskoopin näytöllä. Kaikki eivät kuitenkaan pidä sitä ensimmäisenä videopelinä, koska laite oli täysin mekaaninen, ei sisältänyt tietokone generoitua grafiikkaa ja ei käyttänyt tietokonetta tai muuta muistilaitetta pelin näyttämiseen ja pelaamiseen.

Viisi vuotta myöhemmin Alexander Sandy Douglas kehitti OXO:n ja kuusi vuotta sen jälkeen Willy Higinbotham kehitti Tennis for Two:n (Tennis kahdelle). Molempia voidaan pitää ensimmäisenä videopelinä, mutta kumpikaan ei olisi olemassa ilman Thomas T. Goldsmith Jr. ja Estle Ray Mann löydöksiä ja teknologiaa. (Cohen, Cathode-Ray Tube Amusement Device: The First Electronic Game, 2019)

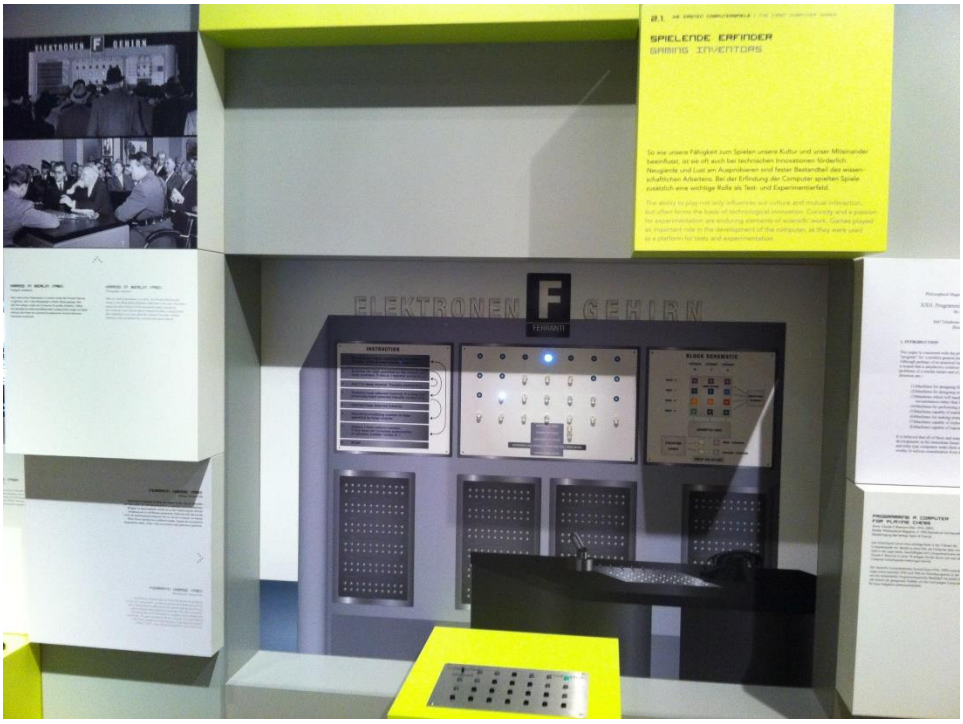
2.3 **Videopelien historia — 1962**

2.3.1 Bertie the Brain (1950)

Kuva 2. Bertie the Brain (Wikipedia 2017)

Bertie the Brain oli ensimmäinen tietokone, mikä oli ohjelmoitu pelaamaan videopelejä. Sen kehitti Josef Kates 1950 Torontossa. Se oli näytillä Kanadian kansallisessa näyttelyssä ja näyttelyn osanottajat pystyivät pelaamaan ristinollaa konetta vastaan. (Ottawa Citizen, 1975)

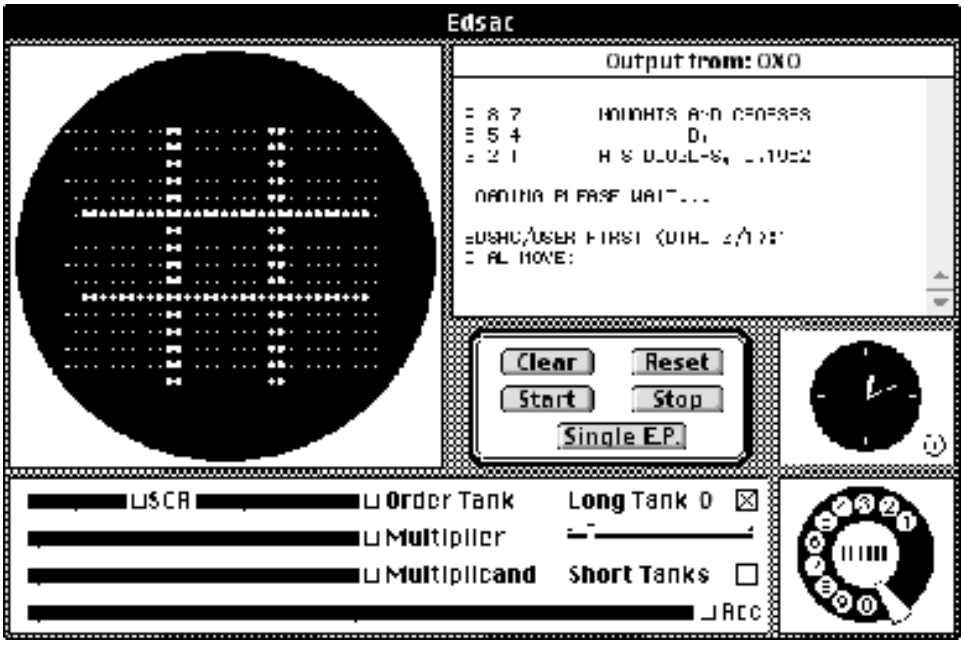
2.3.2 Nimrod (1951)



Kuva 3. Nimrod (Wikipedia 2011)

Nimrod on Ferrantin rakentama tietokone, joka oli suunniteltu pelaamaan videopelejä. Laitteen suunnitteli John Makepeace Bennett ja rakensi Raymond Stuart-Williams. Se oli näytillä Festival of Britain tapahtumassa ja osanottajat pystyivät pelaamaan Nim-nimistä matemaattisstrategiapeliä. Laite oli näytillä myös Berliinin teollisuusnäyttelyssä, jonka jälkeen se purettiin. (Nimrod (computer), n.d)

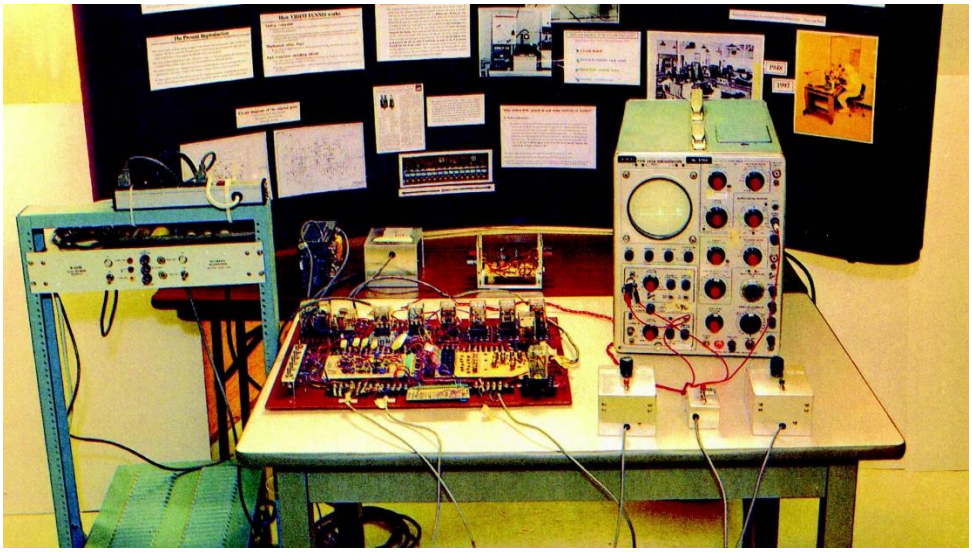
2.3.3 OXO (1952)



Kuva 4. OXO (Wikipedia 2017)

OXO on Electronic Delay Storage Automatic Calculator (EDSAC) kehittämä videopeli. Pelin kehitti ja ohjelmoi Alexander Sandy Douglas. OXO on digitaalinen versio ristinollasta. OXO sisälsi myös ensimmäisen todellisen esimerkin AI:sta. Tietokone ei seurannut valmiiksi ohjelmoituja toimintoja, vaan valitsi siirtonsa pelaajan toimintojen mukaan. (Cohen, OXO aka Noughts and Crosses - The First Video Game, 2019)

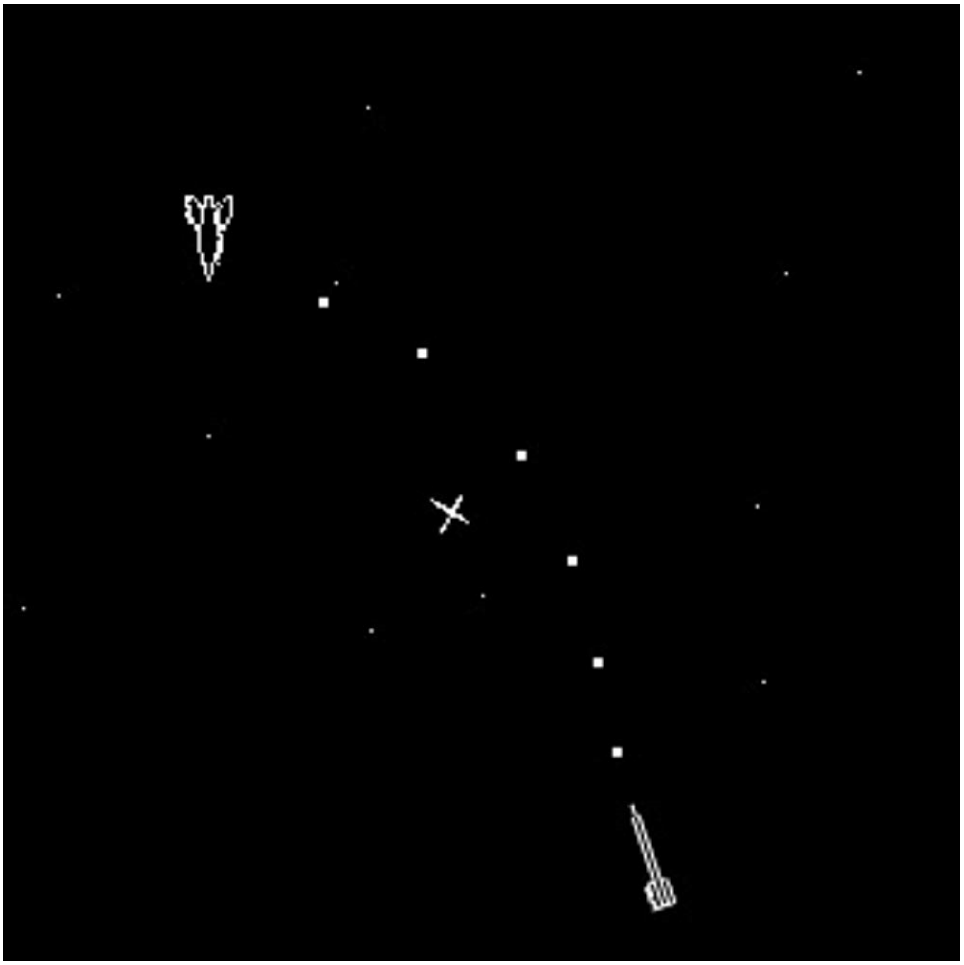
2.3.4 Tennis for Two (1958)



Kuva 5. Tennis for Two (Wikipedia 2013)

Tennis for Two on William Higinbotham kehittämä videopeli, jossa pelaajat pelasivat tennistä. Peli sisälsi kaksi ohjauslaitetta, joita käyttämällä pelaajat syöttivät vihreää valopalloa symbolisen verkon yli. Peli pelattiin oskilloskoopin näytöllä. Peli oli näytillä Brookhavenin kansallisessa laboratoriossa 18. lokakuuta 1958 ja sen suosio ylitti Dr. Higinbothamin odotukset. (Lambert, 2008)

2.3.5 Spacewar! (1962)



Kuva 6. Spacewar! (imdb n.d)

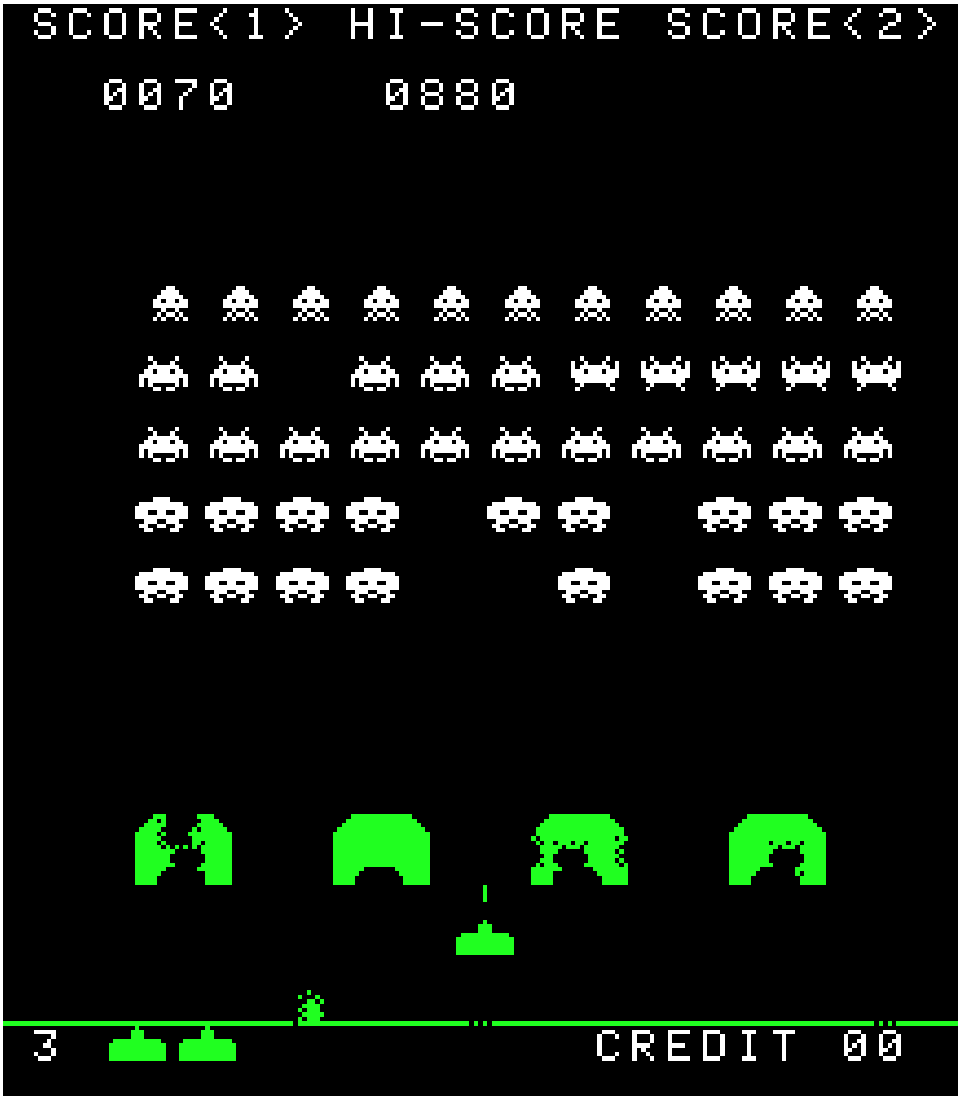
Spacewar! on Steve Russell kehittämä videopeli, missä kaksi pelaajaa taistelevat toisiaan vastaan avaruusaluksissa. Peli luotiin Digital Equipment Corporation (DEC) Programmed Data Processor-1 (PDP-1) laitteelle. Spacewar! oli myös ensimmäinen videopeli, mikä on asennettu useampaan eri tietokoneeseen. Pelissä kaksi avaruusalusta the needle ja the wedge taistelevat toisiaan vastaan tähden painovoiman alaisuudessa. Aluksissa on rajallinen määrä polttoainetta ja torpedoja ja alusten liike noudattaa Newtonin lakeja, joten alukset jatkavat liikettään, vaikka pelaaja ei tekisi mitään. (Spacewar!, n.d)

2.4 **Shoot ’em up pelien historia — 1990**

2.4.1 Mikä on Shoot ‘em up, shmup peli

Shmup-pelejä yhdistää se, että niissä pelaaja taistelee isoa vihollismäärää vastaan ja samalla väistelee näiden ammuksia. Pelaajan menestyminen riippuu hänen reflekseistään pääasiallisesti. Muista genre-elementeistä ei ole yhtä yksimielistä mielipidettä. Osa vaatii, että pelaajan tulee olla jonkinlaisessa aluksessa, kun taas toiset sisällyttävät pelejä, missä pelaaja matkustaa jalan. Shmupit joissa pelaaja liikkuu ilman jonkinlaista kulkuneuvoa voivat sisältää myös pelejä, mitkä eivät kulje raiteilla, toisin kuin aluksessa pelattavat shmupit yleensä.

2.4.2 Space Invaders (1978)



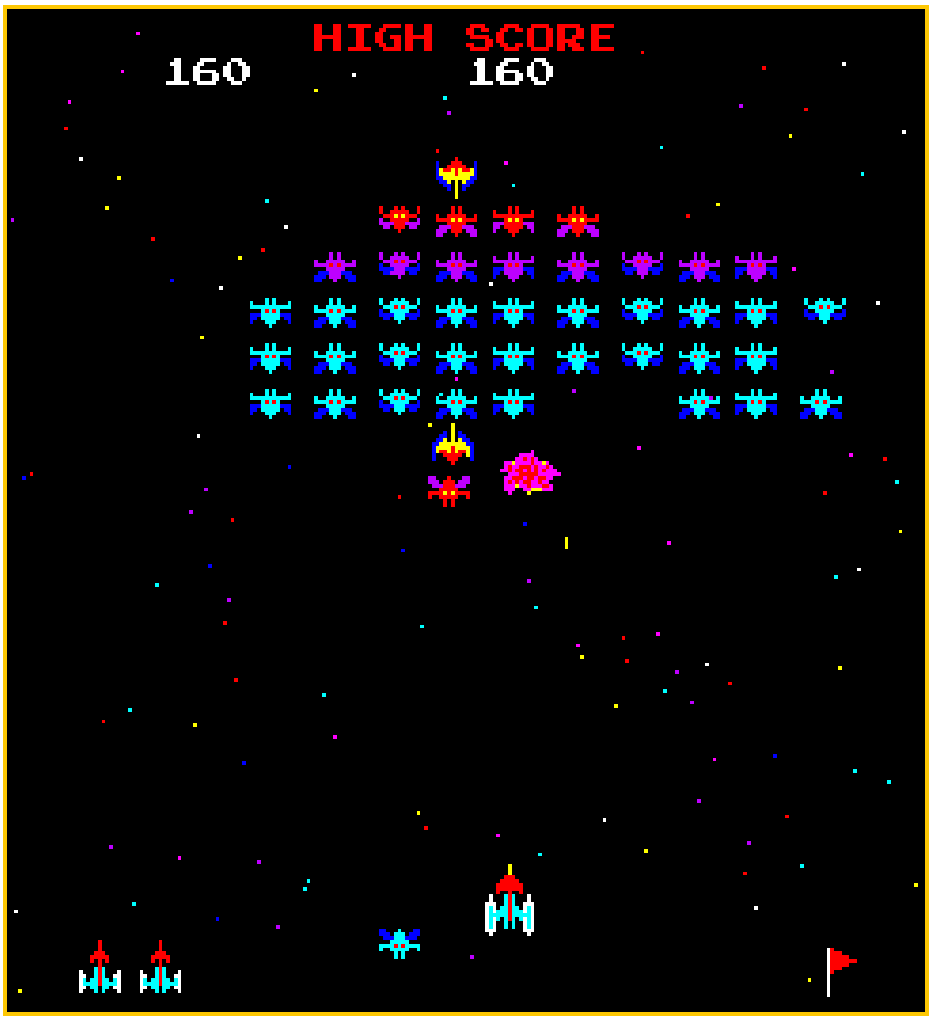
Kuva 7. Space Invaders (Wikipedia 2008)

Space Invadersin kehitti Tomohiro Nishikado vuonna 1978 ja julkaisi Taito. Peli otti inspiraatiota muusta mediasta, kuten The War of the World kirja ja ensimmäinen Star Wars elokuva. Pelistä tuli heti julkaisun jälkeen massiivinen hitti. Pelin suosion noustessa se lisensoitiin USA:n markkinoille 1980. Pelin päämäärä on tuhota kaikki avaruusolennot. Ne liikkuvat määrätyssä muodostelmassa pelaajaa kohti. Avaruusolennot liikkuivat yhtenä kappaleena, koska sen ajan tietokoneet eivät pystyneet liikuttamaan avaruusolentoja erikseen. Pelaajan tulee myös väistellä vihollisten ammuksia ja siinä auttavat kolme kilpeä, mitkä on asetettu pelialueelle. Nämä kilvet eivät kuitenkaan kestä ikuisesti vaan hajoavat, kun niihin on osunut tarpeeksi ammuksia.

Space Invaders on myynyt 300 000 kolikkovideopelilaitetta Japaniin ja 60 000 USA:han. Se oli aikansa parhaiten myynyt videopeli ja tuottoisin

viihdetuote. Nykypäivän rahassa Space Invaders on tuottanut yli 13 miljardia dollaria tehden siitä parhaiten tuottaneen videopelin ikinä. (Space Invaders, n.d)

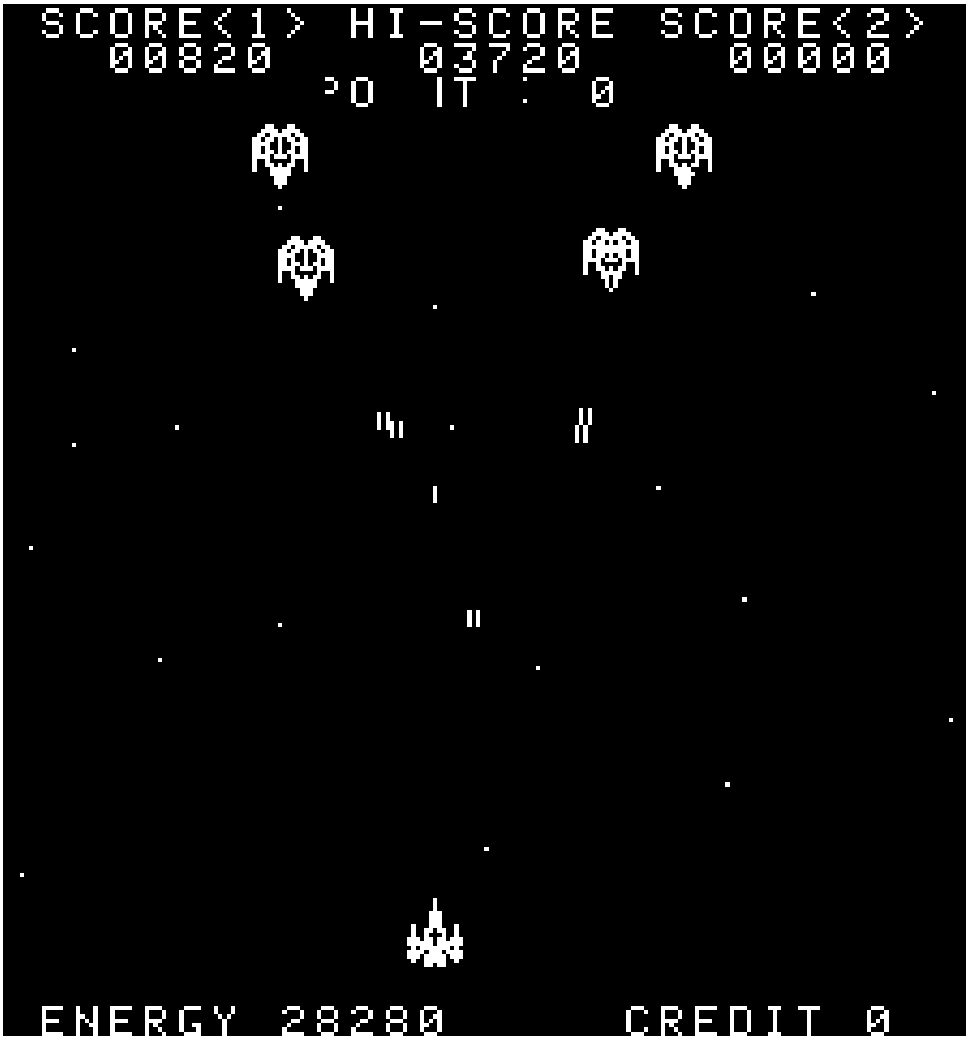
2.4.3 Galaxian (1979)



Kuva 8. Galaxian (Wikipedia 2007)

Galaxian on Namcon julkaisema peli, mikä kehitettiin kilpailemaan Space Invaderin kanssa. Alkunäkymältä Galaxia näyttää hyvin samanlaiselta, mutta siinä viholliset eivät ole rajoitettu liikkumaan kahdessa suunnassa, vaan ne voivat lähteä irti muodostelmastaan ja syöksyä pelaajaa kohti samalla ampuen. Galaxia ei ollut ensimmäinen värillinen peli, mutta se paransi RGB väri grafiikkaa monivärisillä Spriteillään, räjähdyksillään ja fonteillaan. Peli sisälsi myös grafiikkaa, mikä kertoi, kuinka monta elämää pelaajalla on jäljellä ja millä tasolla tämä on. (Galaxian, n.d)

2.4.4 Ozma Wars (1979)



Kuva 9. Ozma Wars (Wikipedia 2010)

Ozma Wars on Logitec Corp. kehittämä ja Shin Nihon Kikaku (SNK) julkaisema peli. Ozma Wars tunnetaan ensimmäisenä pelinä, missä on eroja eri tasojen välillä. Pelin päämääränä on ohjata avaruusalusta ja puolustaa sitä asteroideja, Ufoja ja komeettoja vastaan. Pelaajalla ei ole erillisiä elämiä vaan pelaajan alus sisältää energiaa, mikä laskee jatkuvasti. Vihollisosumat laskevat energiaa paljon. Pelaaja saa energiaa takaisin ajoittaan tukialukselta. Pelissä on 3—4 toisistaan erottuvaa tasoa. Tätä sykliä toistetaan loputtomiin. (Ozma Wars, n.d)

2.4.5 Defender (1981)

Kuva 10. Defender (Wikipedia 2009)

Defender on Williams Electronic:n kehittämä ja julkaisema peli, missä pelaaja taistelee avaruusolentoja vastaan ja puolustaa astronautteja. Pelaaja pystyy liikuttaman alustaan joko oikealle tai vasemmalle. Avaruusolennot yrittävät kaapata astronautteja planeetan pinnalta ja jos ne onnistuvat palaa kyseinen astronautti mutanttina takaisin ja hyökkää pelaajan kimppuun. Taso loppuu, kun pelaaja tuhoaa kaikki avaruusolennot. Jos kaikki astronautit kaapataan planeetta räjähtää ja täyttyy mutanteilta. Jos pelaaja selviää mutantteja vastaan, planeetta korjautuu. Defender on yksi parhaiten myynneistä arkadipeleistä. Sitä on myyty yli 55 000 laitetta. (Defender (1981 video game), n.d)

2.4.6 Scramble (1981)

Kuva 11. Scramble (arcade-museum n.d)

Scramble on Konamin kehittämä ja julkaisema peli, missä pelaaja ohjaa avaruusalusta ja hänen tehtävänänsä on taistella kuuden eri tason läpi tuhoten mahdollisimman monta polttoaine kanisteria ja muita maassa olevia rakennelmia. Peli loppuu, kun pelaaja pääsee tasot läpi ja tuhoaa vihollisten tukikohdan. Pelaajalla on kaksi asetta. Eteenpäin ampuva laser ja alaspäin ampuvat pommit. Pommeilla on tarkoitus tuhota maarakennelmia ja kerätä polttoainetta. Aluksen polttoaine käyrä laskee koko ajan, joten tämä on tärkeää. Pelaaja myös menettää yhden elämänsä yhdestä vihollisosumasta. (Arcade-history, n.d)

2.4.7 Xevious (1982)

Kuva 12. Xevious (Wikipedia 2007)

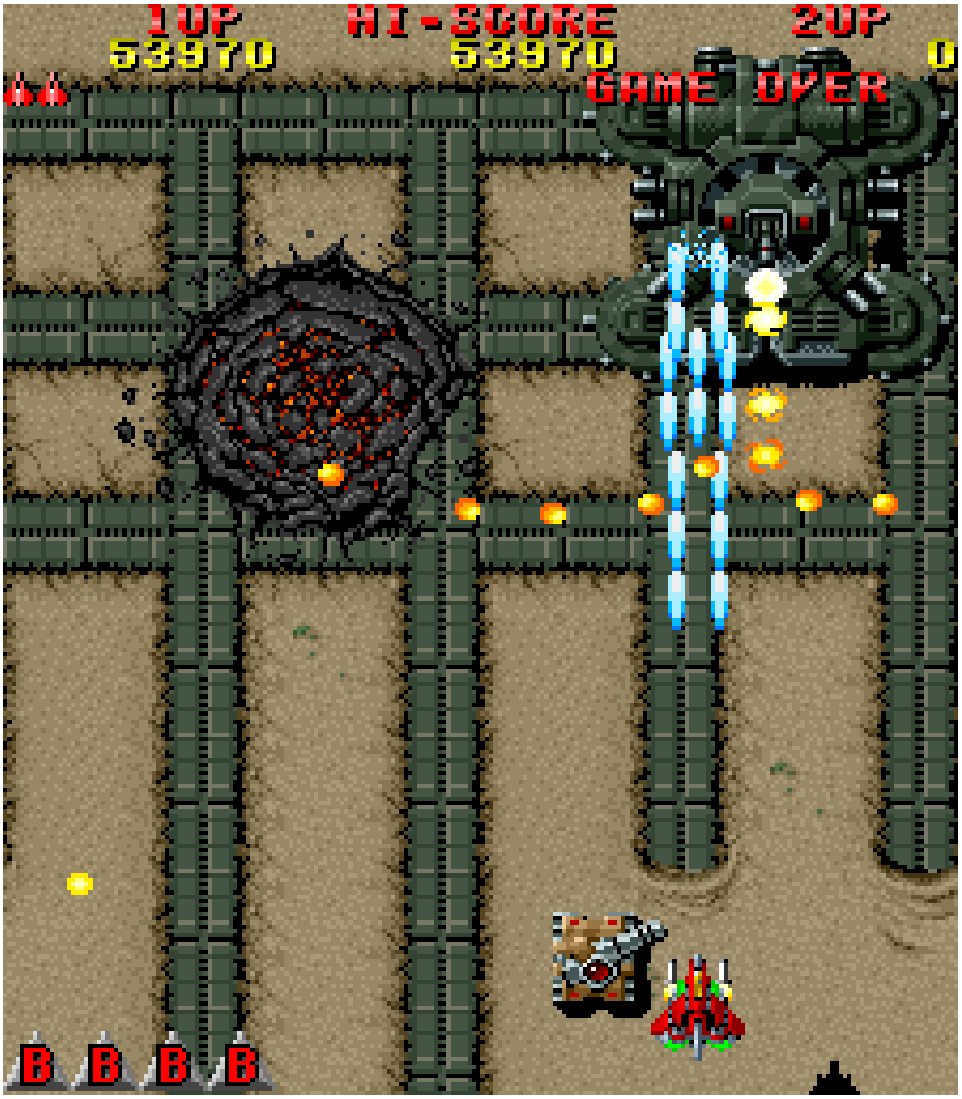
Xevious on Namcon kehittämä ja julkaisema peli, missä pelaaja ohjaa lentokonetta, jossa on eteenpäin ampuva ase ja maahan tiputettavia pommeja. Peli on jaettu 16 eri tasoon ja jos pelaaja kuolee ennen kuin hän on selvinnyt yli 70% tasosta aloittaa hän tason alusta. Jos pelaaja kuolee tämän jälkeen aloittaa hän seuraavan tason alusta. Kun pelaaja läpäisee tason 16 alkaa peli heti uudestaan tasolta 7. Xevious oli ensimmäinen peli, mikä käytti valmiiksi luotuja grafiikoita. Xevious oli myös ensimmäisiä pelejä, joita mainostettiin TV:ssä. (Xevious, n.d)

2.4.8 Gradius (1985)

Kuva 13. Gradius (Wikipedia 2007)

Gradius on Konamin kehittämä ja julkaisema peli, missä pelaajan tulee puolustaa itseään erilaisia avaruusolennoilta. Peli käyttää power meter nimistä järjestelmää, missä pelaaja kerää kapseleita, joita voi sitten käyttää uusien aseiden ostamiseen. Kun Gradius julkaistiin Japanin ulkopuolella, vaihdettiin sen nimeksi Nemesis. Gradiuksesta kehittyi synonyymi sanonnalle Destroy the core!, koska monet Gradiuksen Bossi taistelut perustuivat tiettyjen heikkojen kohtien tuhoamiseen, jotka oli yleensä ilmaistu sinisillä palloilla. (Gradius (video game), n.d)

2.4.9 Raiden (1990)



Kuva 14. Raiden (Wikipedia 2018)

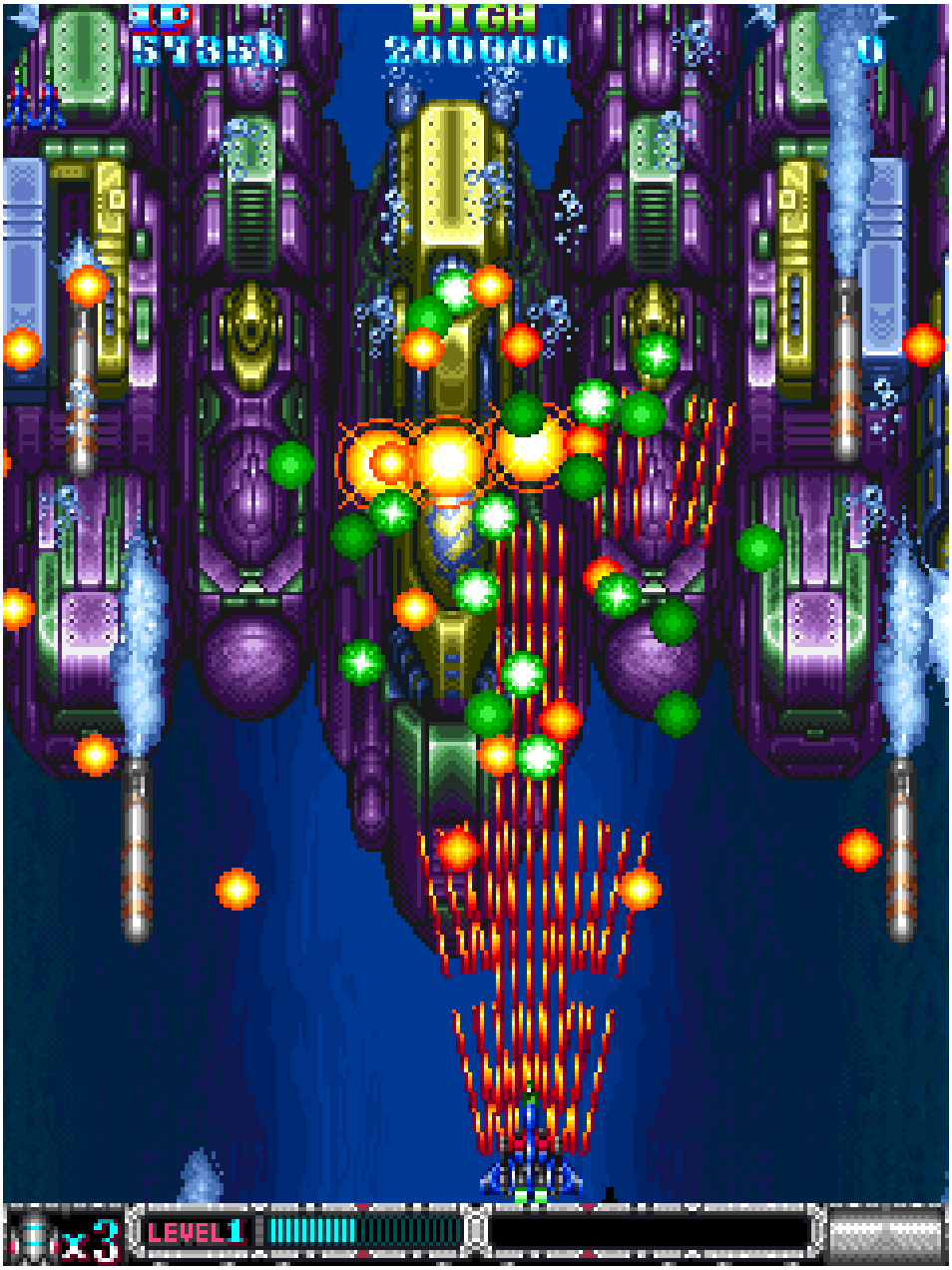
Raiden on Seibu Kaihatsun kehittämä ja Tecmon julkaisema peli, missä pelaajan tehtävänä on puolustaa maapalloa sitä uhkaavaa avaruusolentojen hyökkäystä vastaan. Peli sisältää paljon erilaisia power-up esineitä, mitkä parantavat pelaajan aseita. Pelissä on 8 tasoa ja kun pelaaja läpäisee tason 8 peli on läpäisty ja peli alkaa alusta, mutta viholliset ampuvat nyt nopeammin ja enemmän. Pelaaja saa jokaisesta ylimääräisestä läpäisystä miljoona pistettä lisää. (Raiden (video game), n.d)

2.5 **Bullet Hell pelien historia**

2.5.1 Mikä on Bullet Hell peli

Bullet Hell kuuluu Shoot ‘em up -peleihin. Genre alkoi Japanissa ja siitä on käytetty nimitystä (弾幕) danmaku minkä kirjaimellinen käännös tarkoittaa luotiverhoa. Luotiverhosta voidaan jo tehdä hyviä johtopäätöksiä siitä minkälaiset pelit kuuluvat Bullet Hell -genren alaisuuteen. Bullet Hell -pelien yleisiin ominaisuuksiin kuuluu, että pelaaja kuolee yhdestä osumasta. Kun tämä yhdistetään siihen, että ruudulla saattaa olla parhaimmillaan satoja tai jopa tuhansia ammuksia tekee se Bullet Hell -peleistä hyvin haastavia pelata. Yleiseksi on muodostunut myös se, että viholliset ampuvat kuvioiden muodossa eivätkä vain pelaajaa kohti. Erityisesti bossi-vihollisten ammuskuviot voivat olla todella hienoja ja monimutkaisia. Pelaajalle on yleensä annettu normaalin aseensa lisäksi niin kutsuttu paniikki nappula, jota painamalla pelaaja ampuu yleensä pommin, joka tuhoaa ruudulla olevat ammukset ja tekee vahinkoa ruudulla oleviin vihollisiin. Pelaaja on myös usein kuolematon tämän efektin ajan.

2.5.2 Batsugun (1993)



Kuva 15. Batsugun (Wikipedia 2018)

Batsugun on Toaplanin kehittämä peli. Peli on hyvin tyypillinen shoot 'em up -peli. Pelaaja ohjaa alusta, pelaajalla on eteenpäin ampuva ase ja joka suuntaan räjähtävä pommi, tarkoituksena on tuhota viholliset ja väistellä näiden ammukset. Pelaaja saa tuhotuilta vihollisilta exp-pisteitä. Joka 288. piste nostaa pelaajan aseen tasoa yhdellä. Aseen taso voi nousta vain kaksi kertaa, joten ylimääräiset exp-pisteet antavat pelaajalle lisää pommeja. Pelaaja voi myös kerä P- ja B -ikoneita. P nostaa tämän hetkisen asetason voimaa ja B antaa lisää pommeja (max 8). Batsugunia pidetään ensimmäisenä Bullet Hell -pelinä. (Arcade-history, n.d)

2.5.3 Touhou Project (1996—)

Kuva 16. Touhou 7 – Perfect Cherry Blossom (Wikipedia 2010)

Touhou project on sarja Bullet Hell -pelejä. Pelit on kehittänyt Team Shanghai Alice. Tiimissä on vain yksi jäsen 'ZUN', joka ohjelmoi pelit, tuottaa grafiikat ja säveltää musiikit. Touhou Project lisättiin Guinness World Records:n 2010 tuottoisimpana fanipohjaisena pelisarjana. Sarja on myös synnyttänyt paljon muuta media tuotosta ympärilleen, kuten kirjoja, musiikkia, novelleja, mangaa ja animea. (Touhou Project, n.d)

2.5.4 Ikaruga (2001)

Kuva 17. Ikaruga (Wikipedia 2018)

Ikaruga on Treasuren kehittämä peli, missä pelaaja taistelee vihollisvaltiota vastaan aluksella, joka pystyy vaihtamaan kahden polaarisuuden välillä, musta tai valkoinen. Tämä polaarisuus mekaniikka on Ikarugan ydin ja peli on suunniteltu kokonaan sen ympärille. Kaikki vihollisten ammukset ovat joko mustia tai valkoisia ja pelaajan voi imeä itseensä saman väriset ammukset. Ikarugassa on viisi tasoa ja kolme eri vaikeusastetta. Pelaaja voi nappia painamalla vaihtaa aluksensa polaarisuuden missä vaiheessa vain. Peli on luotu siten, että pelaajan odotetaan vaihtavan polaarisuuttaan

usein ja se muuttaa pelaamisen standardista shoot ’em up -pelistä enemmän pulmapeliksi. (Ikaruga, n.d)

2.5.5 Crimzon Clover (2011)

Kuva 18. Crimson Clover (store.steampowered 2014)

Crimzon Clover on Yotsubanen kehittämä peli. Pelaaja voi valita kolmen eri aluksen väliltä. Alukset käyttävät samoja kontrolleja, mutta niiden nopeus ja break gaugen täyttyminen ovat erilaisia. Pelaajalla on standardi eteenpäin ampuva ase ja sen lisänä on lock on tyylinen laserase, mikä pitää ensin tähdätä käyttäen aluksen tutkaa ja sitten ase pitää laukaista, missä vaiheessa kaikkiin lukittuihin kohteisiin iskee laserammus. Break gauge täyttyy, kun pelaaja tuhoaa vihollisia ja kun se on täyttynyt ja pelaaja laukaisee pommin, siirtyy alus break moodiin, missä alus on hetken tuhoutumaton ja sen aseen teho kasvaa ja vihollisista saatavat pisteet moninkertaistuvat. Pelaaja voi kasata kaksi break moodia päällekkäin ja saada isoimman mahdollisen tulivoiman ja piste kertoimen. Pelissä on kolme eri vaikeusastetta. (Crimzon Clover, n.d)

3 PELIMOOTTORIT

3.1 Unity

Kuva 19. Unity (Wikipedia 2011)

Unity on Unity Technologiesin kehittämä pelimoottori, jolla voidaan kehittää niin 2D- kuin 3D-pelejä. Unity tukee monia alustoja, kuten PS4, Windows, iOS, Android TV, Facebook Gameroom. Unitystä on saatavilla ilmainen versio. Siitä on myös tarjolla kaksi maksullista versiota: Plus ja Pro. Plus on suunnattu yksittäisille kehittäjille ja Pro studioille.

Unityyn voi ladata sisältöä joko ilmaiseksi tai maksua vastaan Unityn Asset Storesta. Siellä on tarjolla niin malleja, tekstuureja, animaatioita kuin ääniä. Unitylle on paljon eri tasoisia tutoriaaleja, joten siihen sisälle pääseminen on helppoa. (Unity (game engine), n.d)

3.1.1 Plussaa

- Hyvä Cross-platform kehitysympäristö
- Helpompi käytettävä, kuin monet vastaavat alustat
- Halpa aloitusinvestointi

3.1.2 Miinusta

- Osa dokumentaatiosta jäljessä, tai puuttuu kokonaan
- Ei tiimi ystävällinen

Unity käyttää ohjelmointikielenään C++ ja C#. Unityllä on tehty mm. Hearthstone (2014), Cuphead (2017), Ori and the Blind Forest (2015) ja Cities: Skylines (2015).

Kuva 20. Cities: Skylines (store.steampower 2015)

3.2 **Unreal Engine**

Kuva 21. Unreal Engine (Wikipedia n.d)

Unreal Engine on Epic Gamesin kehittämä pelimoottori. Alun perin Unreal-pelille (1998) kehitetty moottori on kehittynyt yhdeksi suosituimmista pelimoottoreista. Unreal Engine on täysin C++ pohjainen moottori suunniteltu parhaimmalle mahdolliselle suorituskyvylle. Unreal Engine tukee monia alustoja, kuten PS4, Nintendo Switch, iOS ja PC. Unreal Enginellä pelien kehittäminen on ilmaista ja jos julkaistu peli tuottaa rahaa Epic games ottaa siitä 5% siivun itselleen. Unreal Enginelle on myös oma kauppansa, mistä voi ostaa valmista sisältöä peliinsä. (Unreal Engine, n.d)

3.2.1 Plussaa

- Avoin lähdekoodi
- Tehokas alusta, millä voi tehdä minkä tahansa tasoisia pelejä
- Paljon valmiita työkaluja samassa paketissa

- Hyvä isoille tiimeille

3.2.2 Miinusta

- gameplay framework vaikea opetella
- Puutteita dokumentaatiossa

Unreal Enginellä on tehty mm. BioShock (2007), Borderlands (2009), Gears of War 4 (2016) ja Fortnite (2017).

Kuva 22. BioShock (store.steampower 2007)

3.3 **Phaser**

Kuva 23. Phaser (Phaser n.d)

Phaser on Photon Stormin luoma HTML5 pohjainen pelimoottori. Sen ohjelmointikielenä toimii JavaScript ja sillä voidaan luoda selaimessa tai mobiilissa toimivia pelejä. Phaser vaatii toimiakseen selaimen, joka tukee <canvas> HTML tagia. Tätä tukeviin selaimiin kuuluvat Chrome, Firefox, IE9+ ja Opera. Phaser on täysin ilmainen ja moottorin kehitys rahoitetaan lahjoituksilla. (Phaser (game framework), n.d)

3.3.1 Plussaa

- Toimii suoraan selaimessa
- Avoin lähdekoodi
- HTML5 framework
- Paljon tutoriaaleja perustoiminnoille
- Helppo aloittaa

3.3.2 Miinusta

- Toimii suoraan selaimessa
- HTML5 framework
- Suorituskyky kärsii isommilla kentillä

Phaserilla on tehty mm. Mini Mr.Driller (2018) ja Fantasy Wars (2018).

Kuva 24. Fantasy Wars (Phaser 2018)

Päädyin itse käyttämään Phaseria pelini toteutukseen, koska minun ei tarvinnut opetella uutta ohjelmointikieltä samalla. Olen opintojeni ohessa oppinut ohjelmoimaan JavaScriptillä ja se on vahvin ohjelmointikieleni. Toinen tekijä valinnassani oli Phaserin käyttö yhdellä opintojaksolla. Olin sen aikana päässyt tutustumaan Phaserin perusominaisuuksiin ja olin tunneilla jo alkanut kehittämään tätä peli ideaani. Se että Phaser toimii selaimessa, oli myös yksi syy, miksi halusin käyttää sitä.

3.4 **Godot**

Kuva 25. Godot (Wikipedia 2015)

Godotin kehityksen aloittivat Juan 'reduz' Linietsky ja Ariel 'punto' Manzur 2007 ja sen lähdekoodi julkaistiin GitHubissa 2014 MIT lisenssin alaisena. Godotin ohjelmointikielenä toimii C#, C++ ja GDScript. Godot tukee monia

alustoja, kuten iOS, Android, Windows, macOS. Godot on täysin ilmainen ja sen kehitys rahoitetaan lahjoituksilla. (Godot (game engine), n.d)

3.4.1 Plussaa

- Avoin lähdekoodi
- Täysin ilmainen

3.4.2 Miinusta

- Vaatii enemmän säätämistä per peli objekti
- Muistin hallinta ei niin kehittynyt
- Pelimoottori on vielä monilta osin kesken

Godotilla on tehty mm. Stereobreak ja Falling Word – Challenge your brain.

Kuva 26. Stereobreak (play.google n.d)

4 PELIN TOTEUTUS

Tässä osiossa käyn läpi itse pelin luomisprosessin alkukonseptista itse valmiiseen versioon. Tämä alkaa peligenren valinnalla ja suunnittelulla, jotta on jotain mistä lähteä ideoimaan itse pelin sisältöä. Käymme läpi ideoitani ja miten ne toteutettiin tai miten ne muuttuivat toteutuksen eri vaiheissa. Tuon myös esille lyhyesti muutamia ideoita, joita en pystynyt tai osannut toteuttaa ja miten se vaikuttivat itse peliin.

4.1 **Suunnittelu**

Alussa en ollut vielä varma millaisen pelin haluan tehdä. Pääasiallisina vaihtoehtoina olivat tasohyppely-, vuoropohjainen strategia- ja Bullet Hell -peli. Tasohyppelypeliä ajattelin, koska se olisi ollut helpoin toteuttaa. Mutta samalla se olisi ollut tylsin peli mitä olisin voinut tehdä, koska en oikein saanut mitään hienoa ideaa siihen ja olisin varmaan tehnyt hyvin tavallisen tasohyppelypelin.

Toisena kandidaattina oli jonkinlainen strategiapeli. Alussa en ollut varma halusinko sen olevan tosiaikainen vai vuoropohjainen, mutta hetken pohdittuani miten voisin toteuttaa sen tosiaikaisesti päädyin siihen, että vuoropohjainen peli olisi varmaan parempi, koska siinä pystyisi helpommin suunnittelemaan sen ominaisuuksia, koska ei tarvitsisi ottaa huomioon sitä, että omat hahmot ja vihollishahmot liikkuvat samalla.

Strategiapeliä pohtiessani minulla kävi myös mielessä, jos tekisin jonkinlaisen roolipelin. Ei mitään suurta, mutta vaikka jonkin boss rush -tyylisen roolipelin, missä pelaajan hahmot eivät liikkuneet taisteluiden välissä jossain kartalla vaan he siirtyisivät jollain animaatiolla taistelusta seuraavaan. Tämä idea ei päässyt hirveän pitkälle, koska se ei ollut aiheena niin kiinnostava.

Viimeisenä ideana ja tämän pelin teemana mietin Bullet Hell -tyylistä peliä. Bullet Hell pelit ovat aina kiinnostaneet minua, vaikka en ikinä olekaan olut mitenkään mahtava niissä. Mutta oman tekeminen vaikutti idealta, missä pääsisin kunnolla tekemään kokeiluja ja ideoimaan mahdollisimman outoja liikeratoja vihollisten luodeille. Mallinani ideoilleni käytin pääasiassa Touhou Project (東方Project) nimisen pelisarjan pelejä, koska ne olivat omia suosikkejani.

4.2 **Ohjelmointi**

Aloitin pelin tekemisen kopioimalla kasan koodiesimerkkejä Phaserin omista koodiesimerkeistä. Aluksi kopioin koodin mikä lataa pelaajahahmon ja liikuttaa sitä nuolinäppäimillä.

```
    sprite = game.add.sprite(400, 200, 'phaser');


    upKey = game.input.keyboard.addKey(Phaser.Keyboard.UP);
    downKey = game.input.keyboard.addKey(Phaser.Keyboard.DOWN);
    leftKey = game.input.keyboard.addKey(Phaser.Keyboard.LEFT);
    rightKey = game.input.keyboard.addKey(Phaser.Keyboard.RIGHT);

    weapon = game.add.weapon(5, 'bullet');

    weapon.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
    weapon.bulletAngleOffset = 90;
    weapon.bulletSpeed = 400;
    weapon.fireRate = 60;

    game.physics.arcade.enable(sprite);
    weapon.trackSprite(sprite, 14, 0);
    fireButton = this.input.keyboard.addKey(Phaser.KeyCode.SPACEBAR);

}

function update() {

    if (upKey.isDown)
    {
        sprite.y--;
    }
    else if (downKey.isDown)
    {
        sprite.y++;
    }

    if (leftKey.isDown)
    {
        sprite.x--;
    }
    else if (rightKey.isDown)
    {
        sprite.x++;
    }
    if (fireButton.isDown)
    {
        weapon.fire();
    }

}
```

Kuva 27. Valmiskoodia pelaajan liikuttamiseen

Sen jälkeen latasin koodin, joka saa pelaajahahmon ampumaan eteenpäin. Samalla testasin ampumisfunktion eri elementtejä. Sain pelaajan ampumaan nopeampaa ja vaihtamaan ammusten kulkunopeutta. Myös ammusten lähtökulmaa pystyy muuttamaan.

Sitten muokkasin hahmon liikkumiskoodia paremmaksi. Kokeilin paria eri tapaa liikuttaa hahmoa, mutta palasin lähes samaan versioon kuin aloittaessa. Muutin hahmon nopeuden muuttamisen vain helpommaksi. Samalla siistin koodia vähän.

```
if (upKey.isDown){
    sprite.y -= 3;
}
else if (downKey.isDown){
    sprite.y += 3;
}
if (leftKey.isDown){
    sprite.x -= 3;
}
else if (rightKey.isDown){
    sprite.x += 3;
}
if (fireButton.isDown){
    weapon.fire();
}
```

Kuva 28. Uusi tapa liikuttaa pelaajaa

Kun hahmo liikkuu ja ampuu, päätin lisätä vihollisia peliin. Kopioin koodiesimerkkejä, jotka loivat vihollisen ja laitoin sen omaan funktioonsa, jota pystyin kutsumaan manuaalisesti pelin ollessa käynnissä. Samalla muutin pelaajahahmon ampumiskoodia. Loin myös koodin, joka katsoo ovatko pelaajan ammukset ja vihollinen kontaktissa, ja jos olivat, niin vihollinen kuolee ja ammus kuolee.

```
aliens = game.add.group();
aliens.enableBody = true;
aliens.physicsBodyType = Phaser.Physics.ARCADE;
```

Kuva 29. Ensimmäiset viholliset

```
function createAliens() {
    var alien = aliens.create(200, 400, 'ufo');
        alien.anchor.setTo(0.5, 0.5);
```

Kuva 30. Vihollisten kutsuminen

Koska muutin pelaajan ampumiskoodia, jouduin luomaan sille oman funktion, jota kutsumalla hahmo ampuu tästä lähin. Löysin esimerkin tälle koodille ja muokkasin sitä itselleni sopivaksi.

```
bullets = game.add.group();
bullets.enableBody = true;
bullets.physicsBodyType = Phaser.Physics.ARCADE;
bullets.createMultiple(30, 'bullet');
bullets.setAll('anchor.x', 0.5);
bullets.setAll('anchor.y', 1);
bullets.setAll('outOfBoundsKill', true);
bullets.setAll('checkWorldBounds', true);
```

Kuva 31. Käytin group:ia pelaajan ampumiseen

```
function fireBullet(){
    if (game.time.now > bulletTime)
{
    //  Grab the first bullet we can from the pool
    var bullet = bullets.getFirstExists(false);

    if (bullet)
    {
        //  And fire it
        bullet.reset(sprite.x, sprite.y + 8);
        bullet.body.velocity.y = -400;
        bulletTime = game.time.now + 70;
    }
}
```

Kuva 32. Kuinka pelaaja ampuu alussa

Seuraavaksi halusin tehdä vihollisen aloituspaikasta satunnaisen ja samalla saada liikettä viholliseen. Tein koodin, mikä synnyttää vihollisen satunnaiseen paikkaan haluamallani alueella ja vihollinen liikkuu pelaajaa kohti automaattisesti.

```
function createAliens() {
    alien = aliens.create(game.rnd.integerInRange(10, 790), -50, 'ufo');
        alien.anchor.setTo(0.5, 0.5);
}
```

Kuva 33. Satunnaisuutta vihollisten luontiin

```
var radians = game.physics.arcade.angleBetween(alien, sprite);
var degrees = radians * (180/Math.PI);
game.physics.arcade.velocityFromAngle(degrees, 100, alien.body.velocity);
```

Kuva 34. Viholliset liikkuvat

Seuraavaksi päätin lisätä vihollisia. Muokkasin vihollisskoodia siten, että se synnyttää automaattisesti haluamani määrän vihollisia.

```
if(aliens.length < 10){
    createAliens();
}
```

Kuva 35. Lisää vihollisia

Seuraavaksi halusin, että viholliset voivat ampua. Käytin tähän samaa koodipohjaa, kuin pelaajan ampuminen. Löysin tähän valmiin esimerkin, missä vihollinen ampui pelaajaa kohti. Käytin tätä ja muokkasin sitä vähän, jotta se sopi paremmin omaan koodiini.

```
function enemyFires () {

    enemyBullet = enemyBullets.getFirstExists(false);

    livingEnemies.length=0;

    aliens.forEachAlive(function(alien){
        livingEnemies.push(alien);
    });

    if (enemyBullet && livingEnemies.length > 0){

        var random=game.rnd.integerInRange(0,livingEnemies.length-1);

        var shooter=livingEnemies[random];

        enemyBullet.reset(shooter.body.x, shooter.body.y);

        game.physics.arcade.moveToObject(enemyBullet,sprite,120);
        firingTimer = game.time.now + 100;
    }
}
```

Kuva 36. Viholliset vastaavat tuleen

Seuraavaksi lisäsin koodin, joka seuraa osuvatko vihollisammukset pelaajaan. Lisäsin siihen myös vähän grafiikka elementtejä. Kun pelaaja kuolee, ruudulle ilmestyy GAME OVER teksti. Samalla lisäsin toiminnon, jossa ruutua klikkaamalla peli alkaa alusta, kun pelaaja kuolee.

```
game.physics.arcade.overlap(bullets, aliens, collisionHandler, null, this);
game.physics.arcade.overlap(enemyBullets, sprite, enemyHitsPlayer, null, this);
game.physics.arcade.overlap(alien, sprite, enemyBodyHitsPlayer, null, this);
```

Kuva 37. Osumien tarkistus

```
function enemyHitsPlayer (sprite,bullet) {

    bullet.kill();
    sprite.kill();
    enemyBullets.callAll('kill');

    stateText.text=" GAME OVER \n Click to restart";
    stateText.visible = true;

    game.input.onTap.addOnce(restart,this);
}
```

Kuva 38. Pelaaja kuolee

Alkuperäinen pelin uudestaan aloituskoodi ei oikein toiminut niin kuin olisin halunnut, joten loin funktion, joka oli lähempänä sitä mitä halusin.

```
function restart () {
    aliens.removeAll();
    createAliens();
    sprite.revive();
    stateText.visible = false;
}
```

Kuva 39. Aloitetaan alusta

Kun minulla oli valmiina pelaajahahmo ja vihollisia pääsin testaamaan miten ne toimivat yhdessä. Pelaajahahmo toimi tässä vaiheessa hyvin. Päätin kuitenkin muokata hahmon liikkumista, koska löysin paremman tavan liikuttaa sitä.

```
sprite.body.velocity.x = 0;
sprite.body.velocity.y = 0;

if (cursors.left.isDown)
{
    sprite.body.velocity.x = -200;
}
else if (cursors.right.isDown)
{
    sprite.body.velocity.x = 200;
}

if (cursors.up.isDown)
{
    sprite.body.velocity.y = -200;
}
else if (cursors.down.isDown)
{
    sprite.body.velocity.y = 200;
}
```

Kuva 40. Taas uusi tapa liikuttaa pelaajaa

Testatessani vihollisskoodiani huomasin, että viholliset eivät ammu niin kuin haluaisin. Muokkasin koodia moneen kertaan, kunnes löysin koodiesimerkin, jonka joku oli tehnyt omaan shoot 'em up -peliinsä. Kopioin kyseisen koodinpätkän itselleni ja sovitin sen omaan koodiini.

```
    var firingDelay = 500;
    alien.bullets = 1;
    alien.lastShot = 0;

alien.update = function() {
    if (game.time.now > bulletTime){
        var enemyBullet = enemyBullets.getFirstExists(false);

    if (enemyBullet &&
        this.alive &&
        this.bullets &&
        this.y > game.width / 8 &&
        game.time.now > firingDelay + this.lastShot){

        this.lastShot = game.time.now;
        //this.bullets--;

        enemyBullet.reset(this.x, this.y);
        enemyBullet.body.velocity.y = 400;
        bulletTime = game.time.now + 70;
    }
    game.physics.arcade.moveToObject(enemyBullet,sprite,120);
    }
};
}
```

Kuva 41. Viholliset ampuvat paremmin

Tällä koodilla sain viholliset ampumaan niin kuin halusin, mutta huomasin, että jos teen kaikki vihollisyksiköiden ampumiskoodit tällä tavalla, siitä

tulee paljon sekavaa koodia ja koska se ei ollut omaa koodiani en ymmärtänyt sen toimintaa kunnolla. Aloin ymmärtämään laina koodia vasta myöhemmin, kun olin jo poistamassa sitä, koska olin löytänyt paremman tavan, jolla viholliset voivat ampua. Rupesin tutkiman Phaserin dokumentaatiota, josko sieltä löytyisi parempi tapa luoda vihollisille ammuksia. Löysin Weapon nimisen luokan ja se oli varta vasten luotu luomaan ammuksia ja ampumaan niitä. Kokeilin sitä pelaajahahmoon, ja se toimi niin hyvin, että päätin vaihtaa pelaajan aseen siihen.

```
bullets = game.add.weapon(30, 'bullet');
bullets.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
bullets.bulletSpeed = 600;
bullets.fireRate = 100;
bullets.bulletAngleOffset = 90;
bullets.trackSprite(sprite, 0, 0);
```

Kuva 42. Pelaajalle uusi kiiltävä ase

Päätin vaihtaa vihollisten aseet myös Weapon:in, mutta siinä ilmeni kaikenlaisia ongelmia, koska yritin liittää omaa koodiani jonkun toisen koodiin. Joten päätin luoda uuden vihollisen, joka ei käytä muualta kopioitua koodia, jotta näen, onko Weapon:in käyttö vihollisilla mahdollista. Sain sen toimimaan, kun vihollisia on vain yksi, mutta en saanut sitä toimimaan, kun vihollisia luotiin useampi samasta funktiosta. Taistelin tämän kanssa jonkin aikaa, mutta päätin palata versioon, mikä toimi ja keskittyä muihin asioihin.

Seuraavaksi testasin Phaserin ajastintoimintoja, jotta saisin luotua vihollisia säännöllisin väliajoin. Koska ajastin toiminnot toimivat hyvin nopeasti päätin lisätä ominaisuuden, mikä laski vihollisten määrää ruudulla ja loi uusia vihollisia tarvittaessa. Koska sain ensimmäisen vihollisfunktion luomaan vihollisia tietyin aika välein, ampumaan ja luomaan uusia vihollisia, jos niiden määrä laski alle määritetyn rajan. Päätin kopioida sen koodin ja luoda sen perusteella uuden vihollisen. Yritin taas käyttää Weaponia tällä vihollisella. Sain Weaponin toimimaan yhdellä vihollisella ja huomasin, että jos asetan Weaponin Fireraten == 0 se pystyy ampumaan useampaan suuntaan samaan aikaan, mutta koska firerate on nolla, vihollinen ampuu loputtoman määrän panoksia, joten niiden väistäminen on mahdotonta. Ratkaisu mihin päädyin aluksi oli luoda jokaiselle ampumissuunnalle oma aseensa. Tämä toimi, mutta loi hirveän määrän copy paste koodia. Tulen myöhemmin muuttamaan tämän paljon paremmaksi, mutta nyt olin hyvin tyytyväinen itseni kanssa, koska sain Weaponin toimimaan yhdellä vihollisella.

Kun minulla oli kaksi vihollispohjaa, jotka toimivat hyvin rupesin suunnittelemaan uusia vihollisia. Päätin käyttää ensimmäisenvihollisen koodia pohjana uusille vihollisille, ja loin vihollisen, joka ei ampunut, mutta seurasi pelaajaa ja jos se osui pelaajaan, tappoi se pelaajan. Seuraavaksi loin vihollisen, joka ampuu pelaajaa kohti ja sen molemmille puolille.

Seuraavaksi päätin palata Weaponin pariin ja selvittää miten saisin käytettyä sitä useammalle saman luokan viholliselle. Kokeilin noin miljoonaa eri konfiguraatiota tuloksetta, parhaimmillaan sain vain yhden vihollisen ryhmästä toimimaan kerralla. Jouduin taas palaamaan vanhaan koodiin, mikä toimi useammalla vihollisella.

Koska Weapon ei suostunut yhteistyöhön kanssani, päätin kokeilla luoda luoteja ympyrä muodostelmassa vihollisen ympärille. Aluksi yritin koodata sitä itse käyttäen Phaserin dokumentaatiota ja JavaScriptin dokumentaatiota. Lopulta useamman googlauksen jälkeen löysin valmiin koodin, mikä loi luoteja hiiren ympärille, kun hiirellä klikattiin ruutua. Kopioin tämän koodin ja sovitin sen siten, että luodit syntyivät vihollisen ympärille ajastimella.

```
var amount, start, step, i, angle, speed;
amount = 12;
start = Math.PI * -1;
step = Math.PI / amount * 2;
i = amount;
while (i > 0) {
    var enemyBullet = enemyBullets.getFirstDead();

    if (enemyBullet) {
        enemyBullet.reset(
        alien.x,
        alien.y
        );

        angle = start + i * step;

        speed = 200;

        enemyBullet.body.velocity.x = Math.cos(angle) * speed;
        enemyBullet.body.velocity.y = Math.sin(angle) * speed;
    }
    i--;
}
```

Kuva 43. Ammuksiin kuviota

Päätin nopeasti antaa pelaajalle toisen aseen, koska kun katselin muita Bullet Hell -pelejä, niissä yleensä pelaajalla oli ase molemmilla puolilla hahmoa.

```
playerweapon = game.add.weapon(30, 'bullet');
playerweapon.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
playerweapon.bulletSpeed = 900;
playerweapon.fireRate = 60;
playerweapon.bulletAngleOffset = 90;
playerweapon.trackSprite(sprite, -10, 0);

playerweapon2 = game.add.weapon(30, 'bullet');
playerweapon2.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
playerweapon2.bulletSpeed = 900;
playerweapon2.fireRate = 60;
playerweapon2.bulletAngleOffset = 90;
playerweapon2.trackSprite(sprite, 10, 0);
```

Kuva 44. Kaksi asetta parempi kuin yksi

Tästä Weaponin käytöstä innostuin taas kokeilemaan sitä vihollisille. Kirjoitin koodin, joka ampuu Weaponilla useampaan suuntaan samaan aikaan ja sovitin sen funktioon, joka loi vain yhden vihollisen, koska silloin se toimii. Tästä valitettavasti tuli paljon turhaa koodia, joten myöhemmin kun löysin paremman tavan toteuttaa tämän niin vaihdoin sen.

```
    alien1Bullet1 = game.add.weapon(200, 'bulletEnemy');
    alien1Bullet1.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
    alien1Bullet1.bulletSpeed = 400;
    alien1Bullet1.fireRate = 300;
    alien1Bullet1.bulletAngleOffset = 90;
    alien1Bullet1.trackSprite(alien1, 0, 0);

    alien1Bullet2 = game.add.weapon(200, 'bulletEnemy');
    alien1Bullet2.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
    alien1Bullet2.bulletSpeed = 400;
    alien1Bullet2.fireRate = 300;
    alien1Bullet2.bulletAngleOffset = 90;
    alien1Bullet2.trackSprite(alien1, 0, 0);

    alien1Bullet3 = game.add.weapon(200, 'bulletEnemy');
    alien1Bullet3.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
    alien1Bullet3.bulletSpeed = 400;
    alien1Bullet3.fireRate = 300;
    alien1Bullet3.bulletAngleOffset = 90;
    alien1Bullet3.trackSprite(alien1, 0, 0);

    alien1Bullet4 = game.add.weapon(200, 'bulletEnemy');
    alien1Bullet4.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
    alien1Bullet4.bulletSpeed = 400;
    alien1Bullet4.fireRate = 300;
    alien1Bullet4.bulletAngleOffset = 90;
    alien1Bullet4.trackSprite(alien1, 0, 0);

    alien1Bullet5 = game.add.weapon(200, 'bulletEnemy');
    alien1Bullet5.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
    alien1Bullet5.bulletSpeed = 400;
    alien1Bullet5.fireRate = 300;
    alien1Bullet5.bulletAngleOffset = 90;
    alien1Bullet5.trackSprite(alien1, 0, 0);

alien1.update = function() {
    if(alien1.y > 150){
        alien1.body.velocity.y = 0;
    }
    if(alien1Alive == 1){
    alien1Bullet1.fireAngle = 30;
    alien1Bullet1.fire();
    alien1Bullet2.fireAngle = 60;
    alien1Bullet2.fire();
    alien1Bullet3.fireAngle = 90;
    alien1Bullet3.fire();
    alien1Bullet4.fireAngle = 120;
    alien1Bullet4.fire();
    alien1Bullet5.fireAngle = 150;
    alien1Bullet5.fire();
    }
```

Kuva 45. Ampumiskoodi hirviö

Koska haluaisin, että viholliset voisivat ampua useampaan suuntaan samaan aikaan, olen koittanut muokata sitä vihollisammuntakoodia, mikä toimii useammalla vihollisella sellaiseksi, että viholliset ampuisivat useamman panoksen samaan aikaan, mutta tuloksetta.

Samasta syystä päätin palata nyt varmaan kolmatta kertaa sovittamaan Weaponia toimimaan useammalla saman nimiselle vihollisella samaan aikaan. Ja vihdoin tapahtui ’ihme’ ja sain koodin toimimaan siten, että saman nimiset viholliset pystyivät käyttämään samaa Weaponia ja kaikki

ampuivat oikein. Loin saman tien tällä uudella tiedollani koodin, joka ampuu kohti pelaajaa ja sen molemmille puolille.

```
    var firingDelay = 100;
    alien.lastShot = 0;

    enemyweapon3 = game.add.weapon(600, 'bulletEnemy');
    enemyweapon3.bulletKillType = Phaser.Weapon.KILL_WORLD_BOUNDS;
    enemyweapon3.bulletSpeed = 500;
    enemyweapon3.fireRate = 0;
    enemyweapon3.bulletAngleOffset = 90;

alien.update = function() {
    if (game.time.now > bulletTime){

    if (this.alive &&
        game.time.now > firingDelay + this.lastShot){

        this.lastShot = game.time.now;

        bulletTime = game.time.now + 10;
        enemyweapon3.fire(this, sprite.x, sprite.y);
        enemyweapon3.fire(this, sprite.x-20, sprite.y-20);
        enemyweapon3.fire(this, sprite.x+20, sprite.y+20);
    }
    }
};
```

Kuva 46. Vihollisille uusi kiiltävä ase

Kun sain Weaponin vihdoin yhteistyö haluiseksi päätin ruveta alustamaan pelini ensimmäistä bossi-hahmoa. Phaserilla on valmis toiminto Tween jolla voi luoda valmiiksi määrättyjä liikeratoja. Loin näitä liikeratoja kymmenen ja linkitin ne yhteen, jotta voin keskittyä bossin muihinkin ominaisuuksiin, enkä ole koko ajan muokkaamassa bossin liikeratoja.

```
tween = game.add.tween(boss).to( { x: 400, y: 100 }, 2000, "Linear", false, 1000);
tween2 = game.add.tween(boss).to( { x: 600, y: 200 }, 2000, "Linear", false, 0);
tween3 = game.add.tween(boss).to( { x: 300, y: 100 }, 2000, "Linear", false, 0);
tween4 = game.add.tween(boss).to( { x: 200, y: 100 }, 1000, "Linear", false, 0);
tween5 = game.add.tween(boss).to( { x: 400, y: 200 }, 2000, "Linear", false, 0);
tween6 = game.add.tween(boss).to( { x: 100, y: 300 }, 2000, "Linear", false, 0);
tween7 = game.add.tween(boss).to( { x: 300, y: 100 }, 2000, "Linear", false, 0);
tween8 = game.add.tween(boss).to( { x: 500, y: 200 }, 2000, "Linear", false, 0);
tween9 = game.add.tween(boss).to( { x: 600, y: 200 }, 1000, "Linear", false, 0);
tween10 = game.add.tween(boss).to( { x: 400, y: 100 }, 2000, "Linear", false, 0);

tween.chain(tween2);
tween2.chain(tween3);
tween3.chain(tween4);
tween4.chain(tween5);
tween5.chain(tween6);
tween6.chain(tween7);
tween7.chain(tween8);
tween8.chain(tween9);
tween9.chain(tween10);
tween10.chain(tween);

tween.start();
```

Kuva 47. Bossiin liikettä

Kehitin bossin ominaisuuksia oman aikansa, mutta en päässyt siinä hirveän pitkälle, koska yritin liimata liian monta ominaisuutta bossiin samaan aikaan. Enkä ollut edes kunnolla testannut kaikkia ideoita yksittäin, että toimivatko ne edes. Sain bossille annettua oman hp poolin ja halusin antaa

muillekin vihollisille omansa. Sain annettua hp:n vihollisille, jotka loin yksittäin, mutta en niille, jotka luotiin samaan aikaan. Koitin löytää ratkaisua hetken, mutta sitten löysin tiedon, että näillä vihollisilla on automaattisesti vain yksi hp, joten päätin vain vähentää pelaajan tekemää vahinkoa näihin vihollisiin, jotta ne eivät kuole liian nopeasti.

```
function collisionHandler(bullets, alien) {
    bullets.kill();
    alien.health-=playerdamage/2;
    console.log(alien.health);
    if(alien.health <= 0){
    alien.kill();
    //aliens.remove(alien);
    }
}
```

Kuva 48. Viholliset eivät enää kuole ensimmäisestä osumasta

Kun sain vihollisten hp:n toimimaan rupesin luomaan uusia vihollisia. Käytin vanhoja vihollisia pohjina ja sitten vain muokkasin uuden vihollisen ampumistapaa. Yritän myös tasapainottaa vanhoja vihollisia paremmiksi, jotta voin luoda useampia vihollistyyppejä samaan aikaan.

Nyt kun minulla on useampi toimiva ampumiskoodi vihollisille, päätin palata ensimmäisen bossin pariin. Liimasin kaikki eri ampumistyylit bossiin ja rupesin soveltamaan niistä vaikeampia versioita. Samalla jaoin bossin eri aseet kolmeen eri vaiheeseen, jotka vaihtuvat riippuen bossin jäljellä olevasta hp:sta. Muokkasin samalla Tweenejä, jotta ne toimivat paremmin kunkin vaiheen aseiden kanssa ja jotta pelaaja pystyy ampumaan bossia helpommin, jotta taistelu ei kestä ikuisuuden, koska pelaaja ei saa bossia kiinni. Samalla huomasin ongelman, jossa Tweenit alkoivat koko ajan alusta uudestaan, joten minun täytyi ympäröidä uudet Tween aloitukset niin, että koodi käy kutsumassa ne vain kerran.

```
if (game.time.now > bulletTime && boss.health < 100 && boss.health >= 50){
    if(phaseChange == 1){
    tween2.loop(false);
    tween2.stop(true);
    phaseChange = 0;
    }
```

Kuva 49. Bossi vaihtaa seuraavaan vaiheeseen

Sitten palasin bossin aseeseen, joka ampui ympyrä muodostelmassa ammuksia joka suuntaan. Halusin sen ampuvan kaksi eri nopeuksista aaltoa ammuksia. Tässä vaiheessa käytin vielä sitä valmis koodia, minkä olin löytänyt tätä varten, mutta koska en saanut sitä sovitettua haluamakseni rupesin tutkimaan, saisinko korvattua sen osuuden, mikä itse ampuu panoksen Weaponilla. Sain Weaponin toimimaan, mutta jouduin kirjoittamaan ympyrä koodin uudestaan.

```
var amount, i, angleSegment;
amount = 36;
angleSegment = 360 / amount;
i = amount;
while (i > 0) {

        bossweapon.fireAngle = angleSegment * i;
        bossweapon.fire(this);
    i--;
}
```

Kuva 50. Uusi kiiltävä ympyrä koodi

Löysin samaan aikaan myös paremman tavan tähdätä vihollisaseet pelaajaa kohti ja pääsin korvaamaan paljon koodia tällä uudella tavalla.

```
var angle = game.physics.arcade.angleToXY(alien, sprite.x, sprite.y);
enemyweapon3.fireAngle = angle * 180 / Math.PI;
enemyweapon3.fire(this);
enemyweapon3.fireAngle = angle * 180 / Math.PI+3;
enemyweapon3.fire(this);
enemyweapon3.fireAngle = angle * 180 / Math.PI-3;
enemyweapon3.fire(this);
```

Kuva 51. Viholliset käyttävät aseitaan paremmin

Kun olin saanut paljon itse pelistä tehtyä, päätin keskittyä hetken peli mukavuuden kehittämiseen. Normaalisti Bullet Hell -tyylisissä peleissä, missä pelaajalla on vain yksi elämäpiste ja useampi elämä pelaajahahmo ottaa vahinkoa vain hyvin pieneksi rajatulla alueella, mutta omalla hahmollani tämä alue on tällä hetkellä koko hahmon alue. Joten päätin liimata pelaajaan kiinni punaisen täplän, joka toimii pelaajan osuma-alueena.

Tämä osoittautui yllättävän vaikeaksi tehtäväksi, koska pelaajahahmo ja punainen täplä eivät suostuneet menemään oikein päin. Ja täplä oli aina pelaajan alla, jolloin sitä ei nähnyt. Jouduin muuttamaan, miten luon nämä molemmat spritet ja muutin koodia siten, että täplä on liimattu pelaajahahmoon ja täplä oli se mitä pelaaja ohjasi ja hahmo vain kulki mukana tekemättä mitään. Samalla muutin kaikki osuman tarkastus koodit siten, että ne seurasivat täplää eivätkä pelaajahahmoa.

```
sprite = game.add.sprite(400, 500, 'character');
sprite.anchor.setTo(0.5, 0.5);
game.physics.arcade.enable(sprite);
sprite.body.collideWorldBounds = true;

sprite2 = game.add.sprite(0, 0, 'phaser');
sprite2.anchor.setTo(0.5, 0.5);

sprite.addChild(sprite2);
```

Kuva 52. Pelaaja oppi väistelemään paremmin

Kun olin saanut useamman vihollistyypin toimimaan ja pelaaja on pääasiallisesti valmis. Päätin luoda funktion, joka kutsuu eri vihollistyyppejä

ennalta määrätyin väliajoin. Aloitin koodin perinteiseen tapaani ja rupesin lisäämään koodiin miljoona if else -lausetta. Tämä oli todella huono tapa luoda tämä koodi ja siirryin nopeasti käyttämään switchia, koska siinä pystyin helposti luomaan useamman tilanteen ja samalla määräämään, milloin nämä tilanteet olivat ja mitä silloin tapahtui. Koodista tuli samalla paljon luettavamman näköistä.

```
function spawn(){
    console.log('loop');
if(start){
    switch(i){
        case(0):
            game.time.events.add(500, createAlien1);
            break;
        case(1):
            game.time.events.add(500, createAlien2);
            break;
        case(2):
            game.time.events.add(500, createAlien3);
            break;
        case(3):
            game.time.events.add(500, createAliens1);
            break;
        case(4):
            game.time.events.add(500, createAliens2);
            break;
        case(5):
            game.time.events.add(500, createAliens3);
            break;
        case(20):
            game.time.events.add(500, firstBoss);
            break;
        default:
            //spawntimer = game.time.now;
            break;
    }
    i++;
}
game.time.events.add(5000, spawn);
}
```

Kuva 53. Vihollisille vuoronumeroita

Samalla päätin hienosäätää vihollisten käyttäytymistä hieman, koska niiden tulisi toimia hyvin niin yksin, kuin yhdessä. Kehitän muutamalle viholliselle liikeratoja, jota pitkin ne sitten liikkuvat. Aluksi yritin käyttää tähän Tweeniä, mutta en saanut sitä toimimaan viholliskoodissa, missä kutsuin useamman saman nimisen vihollisen samaan aikaan. Joten jouduin tekemään muutaman if else koodin, mitkä eivät olleet mitään hirveän kaunista katsottavaa, mutta ne toimivat niin kuin halusin.

```
alien.update = function() {

    if(this.y > 150 && this.y < 160){
        this.body.velocity.x = -100;
```

Kuva 54. Tavallisiin vihollisiin liikettä

Seuraavaksi muutin vihollisten luontikoodia siten, että osa niistä syntyi eri paikkoihin, mutta kuitenkin niin, että ne syntyivät aina samoihin eri kohtiin, jos koodin kutsui uudestaan.

```
var i = 0;
var spawn = [200,400,300,500,600,700,600,200,300,400];
aliens4();
function aliens4(){
var alien = aliens.getFirstExists(false);
if (alien) {
    alien.reset(spawn[i], -50);
    alien.body.velocity.x = 0;
    alien.body.velocity.y = 100;
```

Kuva 55. Viholliset samoihin eri paikkoihin

Seuraavaksi kopion yhden viholliskoodeistani, ja muutin niitä siten, että toinen kutsui vihollisia pelialueen vasempaan puoliskoon ja toinen loi niitä oikeaan puoliskoon.

Tässä vaiheessa peli on hyvällä mallilla ja kaikki sen osat toimivat suurin piirtein tahdotuilla tavoilla. Joten päätin optimoida koodia, jotta se olisi selkeämpää ja helpommin muokattavissa. Samalla lisäsin bossiin vielä pari uutta asetta, jonka jälkeen rupesin luonnostelemaan bossin hp:n visuaalista esittämistä. Halusin bossin hp:n näkyvän pelialueen ylälaidassa ja pienentyvän visuaalisesti samalla kun bossi ottaa vahinkoa.

```
bosshealth = 2000;
healthBar = game.add.sprite(20, 20, 'healthBar');
healthBar.cropEnabled = true;
boss = game.add.sprite(400, -50, 'bossUfo');
boss.health = bosshealth;
boss.anchor.setTo(0.5, 0.5);
boss.scale.setTo(3, 3);
game.physics.arcade.enable(boss);
console.log(healthBar.width);
healthBar.width = (boss.health / bosshealth) * 750;
```

Kuva 56. Bossin elämä näkyy kaikille

Muutin samalla bossin koodia siten, että sen eri vaiheet alkoivat, kun bossin hp on alle tietyn rajan.

```
}
if (game.time.now > bulletTime && boss.health > bosshealth * 0.3 && boss.health <= bosshealth * 0.6){
    if(phaseChange == 1){
    tween2.loop(false);
    tween2.stop(true);
    phaseChange = 0;
    }
```

Kuva 57. Paranneltu versio bossin vaiheen vaihdosta

Vihollisia on mielestäni tehty hyvä määrä, niin voin luoda varsinaisen peli tason, jotta pääsen testaamaan sitä. Suunnittelin, mitkä viholliset syntyvät mihinkin aikaan ja mitkä syntyvät samaan aikaan, jotta peli tuntuu haastavalta, mutta ei liian vaikealta. Tämä koodi sai yllättävän nopeasti viimeisen

muotonsa, lukuun ottamatta sitä kuinka paljon vihollistyyppejä on ruudulla samaan aikaan.

```
case(1):
    game.time.events.add(500, createAliens1);   //Random circle spawn.
    game.time.events.add(500, createAliens3);   //Homing trible shot horizontal.
    break;
case(2):
case(!alien1Alive):
    game.time.events.add(500, createAlien1);    //Slow circle shot.
    game.time.events.add(500, createAliens1);   //Random circle spawn.
    break;
case(3):
    game.time.events.add(500, createAliens4);   //multi single circle shot left.
    game.time.events.add(500, createAliens5);   //multi single circle shot right.
    game.time.events.add(500, createAliens1);   //Random circle spawn.
    break;
case(4):
case(!alien2Alive):
    game.time.events.add(500, createAlien2);    //Fast circle shot.
    game.time.events.add(500, createAliens2);   //Homing trible shot vertical.
    break;
case(5):
case(!alien1Alive):
    game.time.events.add(500, createAliens4);   //multi single circle shot left.
    game.time.events.add(500, createAliens5);   //multi single circle shot right.
    game.time.events.add(500, createAlien1);    //Slow circle shot.
    break;
case(6):
```

Kuva 58. Ryhmä passeja vihollisille

Lisäsin seuraavaksi mekaniikan, joka tappaa pelaajan, jos se ottaa liikaa vahinkoa. Ja samalla loin koodin, joka ilmoittaa, että peli on hävitty ja klikkaamalla pelaaja pääsee takaisin alkuun.

```
function enemyHitsPlayer (sprite,bullets) {

    bullets.kill();
    playerHealth--;
    if(playerHealth <= 0){
        sprite.kill();

    stateText.text=" GAME OVER \n Click to restart";
    stateText.visible = true;
    game.input.onTap.addOnce(restart,this);
    }
}
```

Kuva 59. Peli ei lopu enää yhdestä osumasta

```
function restart () {
    game.state.restart();
}
```

Kuva 60. Parempi tapa aloittaa peli uudestaan

Seuraavaksi muutin vihollisten ampumiskoodia siten, että voin muuttaa vihollisten ammusten nopeutta toisistaan riippumatta vaikka muutama vihollinen käyttää samaa Weapon luokkaa ampumiseen.

```
bulletTime = game.time.now + 10;
var angle = game.physics.arcade.angleToXY(alien, sprite2.x, sprite2.y);

enemyweapon.bulletSpeed = 100;
enemyweapon.bulletSpeedVariance = 0;
enemyweapon.fireAngle = angle * 180 / Math.PI;
enemyweapon.fire(this);
enemyweapon.fireAngle = angle * 180 / Math.PI+5;
```

Kuva 61. Ammukset vaihtavat nopeuttaan

Sen jälkeen lisäsin järkevämmän aloituksen pelille, koska tähän mennessä olen aloittanut sen vain kutsumalla funktiota, mikä aloittaa pelin. Lisäsin aloitusruutuun napin, jota painamalla peli alkaa. Samalla tyylittelin napin niin, että se ei jää häiritsemään pelaamista.

```
button = game.add.button(game.world.centerX - 95, 300, 'button', gameplay, this, 2, 1, 0);
```

Kuva 62. Nappi aloitukseen

Seuraavaksi päätin optimoida koodia toimivaan paremmin, koska pelillä oli tapana hidastua mitä pidemmälle siinä mentiin. Selvitin, mitkä koodin pätkät eivät loppuneet silloin kun niiden pitäisi ja korjasin löydöksiäni parhaani mukaan. Yksi isompi ongelma, minkä huomasin, oli että kun olen kutsunut jonkun kolmesta isommasta vihollisestani niin niiden koodi ei loppunut, vaikka kyseinen vihollinen kuoli. Selvitin myös, miten Phaserin eri osien poisto koodit toimivat ja löytyykö sieltä syitä, miksi peli hidastuu. Phaserillä on kaksi eri tapaa poistaa tavaraa peli alueelta. Kill() ja destroy(). Näiden tapojen erona on se, että kill() jättää tappamansa objektin tietyt osat silti eloon. Se vain keskeyttää sillä hetkellä sen toiminnan, mutta se jää taustalle viemään tilaa. Olin tähän mennessä käyttänyt kaiken tappamiseen kill() komentoa, mutta päätin siirtyä käyttämään destroy() komentoa, koska se tuhoaa koko objektin saman tien eikä jätä jälkeensä ruumiita, mitkä kasautuisivat pelin aikana ja hidastaisivat peliä. Tämä loi oman ongelmansa, koska destroy() tappaa kohteensa saman tien, eikä jätä mitään jälkeensä, se aiheutti paljon ongelmia muiden koodin osien kanssa, mitkä tarvitsivat näitä objekteja toimiakseen, joten minun piti ympäröidä nämä koodin osat niin, että jos niiden tarvitsema objekti kuoli, niin tämä koodi ei pystynyt jatkamaan alueelle, mikä olisi aiheuttanut ongelmia.

```
function collision2Handler(alien2, bullets){
    bullets.kill();
    alien2.health-=playerdamage;
    if(alien2.health <= 0){
    alien2Alive = 0;
    game.time.events.add(500, removeSpriteEnemy2);
    alien2.kill();
    }
}
function removeSpriteEnemy2(){
    if(!alien2Alive){
        alien2.destroy();
    }
}
```

Kuva 63. Vihollisen poisto, kun se kuolee

Tähän mennessä olin käyttänyt pelin aloittamiseen ja resetointiin saman näköistä nappia, minkä olin vain löytänyt netistä, mutta seuraavaksi tein itselleni kaksi nappia, jotka kertovat paremmin, mitä ne tekevät. Samalla tein niille molemmille erilliset kutsumiskoodit, jotta ne eivät sekoitu keskenänsä.

```
button = game.add.button(game.world.centerX - 95, 300, 'start', gameplay);
button2 = game.add.button(game.world.centerX - 95, 400, 'restart', restart);
button2.visible = false;
```

Kuva 64. Toinen nappi lopetukseen

Tässä välissä nopeasti muokkasin vihollisten ampumista mieluisammaksi ja sen jälkeen lisäsin kuvan, joka selittää pelin kontrollit pelaajalle.

```
controls = game.add.sprite(30, 30, 'controls');
controls.scale.setTo(0.5, 0.5);
```

Kuva 65. Opaste kyltti pelaajalle

Sitten loin muuttujan, joka seuraa onko peli alkanut ja sallii tai kieltää halutut toiminnot pelaajalta riippuen sen tilasta. Tähän mennessä, kun pelaaja kuoli, niin koodi vain jatkoi kulkuaan, mutta nyt saan pysäytettyä sen haluamaani kohtaan. Samalla muokkasin vähän vihollisten luonnin ajoitusta.

```
function gameplay(){
    start = 1;
    var i = 0;
    button.visible = false;
    controls.visible = false;
    gamestart = 1;
    spawn();

    function spawn(){
    if(start){
    if(gamestart){
        switch(i){
            case(0):
```

Kuva 66. Peli alkaa ja loppuu kun käsketään

Seuraavaksi taas vähän vihollisten ampumisen optimointia. Ja muokkaan, miten yksi vihollinen ampuu uudestaan, jotta saan ampumisen näyttämään paremmilta.

Halusin yhden vihollisistani ampuvan kaksi eri nopeuksista ympyräkuviota, joten kopioin alkuperäisen ympyräkuviokoodin ja liitin sen samaan koodiin eri nimellä ja vaihdoin sen ampuma nopeutta mieluisaksi. Muutin myös molempien koodien tuottamien ammusten määrää ja päädyin versioon, missä hitaampi ympyrä sisältää puolet vähemmän ammuksia kuin nopeampi. Testasin lopuksi vielä lisätä kolmannen ympyrän, mutta se oli omasta mielestäni liikaa.

```
var amount, i, angleSegment;
var amount2, j, angleSegment2;
var amount3, k, angleSegment3;
    amount = 24;
    angleSegment = 360 / amount;
    i = amount;
    while (i > 0) {

            enemyweapon2.bulletSpeed = 100;
            enemyweapon2.fireAngle = angleSegment * i;
            enemyweapon2.fire();

        i--;
    }
    amount2 = 48;
    angleSegment2 = 360 / amount2;
    j = amount2;
    while (j > 0) {

            enemyweapon2.bulletSpeed = 150;
            enemyweapon2.fireAngle = angleSegment2 * j;
            enemyweapon2.fire();

        j--;
    }
    amount3 = 12;
    angleSegment3 = 360 / amount3;
    k = amount3;
    while (k > 0) {

            enemyweapon2.bulletSpeed = 200;
            enemyweapon2.fireAngle = angleSegment3 * k;
            enemyweapon2.fire();

        k--;
    }
```

Kuva 67. Lisää ympyröitä

Minulla on myös toinen vihollinen, joka ampuu ympyräkuviolla ja halusin tehdä siitä vaikeamman. Päädyin ratkaisuun, missä ympyrä kiertää vihollisen ympärillä, jolloin se ampuu ammukset aina vähän eri paikkaan. Samalla muutin sen ampumanopeutta, koska ammusten väistely oli vähän liian vaikeaa vanhalla nopeudella ja ottaen huomioon, että samalla pelaajan tulisi pystyä väistelemään useamman vihollisen ammuksia.

```
var angle;
    angle = 0;
    while (angle < 360) {

            enemyweapon3.bulletSpeed = 100;
            enemyweapon3.fireAngle = angle + offset;
            enemyweapon3.fire();
        angle+= 10;
    }
    offset+= 3;
    if(offset >= 10){
        offset = 0;
    }
```

Kuva 68. Vaikeampi ympyrä

Seuraavana työpöydälläni oli viimeinen yksittäinen vihollinen. Halusin muuttaa sen aseen kokonaan erilaiseksi. Tähän mennessä kyseinen vihollinen on ampunut kolme kuuden ammuksen sarjaa pelaajaa kohti. Reunimmaiset sarjat ammuttiin vähän pelaajan molemmin puolin, ja kaikki kuusi ammusta liikkuivat eri nopeudella. Nyt vihollinen ampuu kaksi jatkuvaa sarjaa ammuksia pelaajan molemmille puolille siten, että niitä ei voi väistää ja pelaaja on jumissa niiden välissä. Samalla vihollinen ampuu yhden ammuksen pelaajaa kohti, jonka pelaaja pystyy väistämään, mutta se on vaikeampaa, koska pelaaja joutuu kiinnittämään huomiotaan sivuilla oleviin ammuksiin.

```
    var angle = game.physics.arcade.angleToXY(alien3, sprite2.x, sprite2.y);
    enemyweapon4.bulletSpeed = 500;
    enemyweapon4.fireAngle = angle * 180 / Math.PI+10;
    enemyweapon4.fire();
    enemyweapon4.fireAngle = angle * 180 / Math.PI-10;
    enemyweapon4.fire();

if(game.time.now > bulletTime){

    enemyweapon4.bulletSpeed = 200;
    enemyweapon4.fireAngle = angle * 180 / Math.PI;
    enemyweapon4.fire();
```

Kuva 69. Luoti verhot

Kun kaikki viholliset bossia lukuun ottamatta on muokattu siten, että ne toimivat hyvin keskenään ja eivät ole liian haastavia, päätin muokata bossia paremmaksi. Bossin aseet ovat periaatteessa samat kuin tavallisilla vihollisilla. Sen versio mistäkin aseesta vain on vaikeampi. Parissa tilanteessa käytän myös useampaa asetta samaan aikaan, jotta saan pakotettua pelaajan liikkumaan jatkuvasti.

Seuraavaksi optimoin bossin eri vaiheita.

- Kuinka nopeasti eri aseet ampuvat.
- Kuinka paljon vahinkoa bossi ottaa pelaajan ammuksista.
- Missä vaiheessa bossin eri vaiheet alkavat.
- Miten bossi liikkuu eri vaiheissa, jotta pelaaja pystyy ampumaan bossia mahdollisimman usein.

Lopuksi vielä tein bossin viimeisen aseen, jonka tarkoituksena oli aluksi ampua täysin satunnaisesti ammuksia, mutta huomasin, että välillä pelaaja kuoli koska ei ollut mahdollista väistää kaikkia ammuksia. Päädyin ratkaisuun, missä ammukset lentävät ennalta määrätyssä kuviossa, mutta tämä kuvio on mahdollisimman monimutkainen.

```
if (game.time.now > bulletTime && boss.health < bosshealth * 0.3 && phaseChange2 == 1){
    tween3.loop(false);
    tween3.stop(true);
    phaseChange2 = 0;
    bossweapon.bulletGravity.y = 100;
    offset = 0;
    bossweapon.killAll();
    }

if (game.time.now > bulletTime && boss.health < bosshealth * 0.3 && this.alive && game.time.now > bulletTime){

    bulletTime = game.time.now + 200;

    angle = 0;
    while (angle < 360) {
        bossweapon.bulletSpeed = 150;
        bossweapon.fireAngle = angle + offset;
        bossweapon.fire(this);
        angle+= 20;
    }
    offset++;
    if(offset <= -5){
        offset = 0;
    }
}
```

Kuva 70. Loppuhuipennus

4.3 **Pelattavuuden hiominen**

Pelin testaaminen oli kokoaikainen osa pelintekoprosessia, koska samalla piti testata, että pelielementit toimivat niin kuin niiden pitää ja samalla testata pystyykö pelaaja selviytymään uudesta uhasta.

Koska Phaser on JavaScript -pohjainen pelimoottori, pelin testaaminen pystyttiin toteuttamaan selaimessa. Käytin tässä apuna Amazonin cloud9 -palvelua. Cloud9 on pilvipohjainen ohjelmointi ympäristö, missä voi koodata, debugata ja ajaa koodia suoraan selaimessa. Asensin Phaserin suoraan cloud9:n päälle, joten pystyin vain ajamaan koodiani suoraan cloud9:sta.

Pelin testaaminen tapahtui kahdessa vaiheessa. Ensimmäisessä vaiheessa tarkoituksena on saada haluttu toiminto toimimaan kuten halutaan, kuten vaikka vihollisen ampumistapa. Alussa ei ole väliä ampuuko vihollinen 10 vai 200 ammusta, kunhan ne kaikki tulee ammuttua ja ne liikkuvat

halutulla tavalla. Tämä on pitkäkestoisin testausvaihe, koska kaikki tekevät virheitä ja niiden löytämiseen menee aina aikaa. Samalla saattaa olla, että mitä tehdään ei olekaan mahdollista tavalla, jota yritetään ja sen huomaamiseen menee oma aikansa. Tämä vaihe voidaan myös tehdä useampaan kertaan samalla koodin pätkälle, koska myöhemmässä vaiheessa ollaan löydetty tapa toteuttaa se paremmin tai selkeämmin.

Testaamisen toisessa vaiheessa tarkoituksena on tasapainottaa pelimekaniikat siten, että ne ovat haastavia, mutta eivät mahdottomia pelaajalle. Tämän pelin kohdalla se tarkoittaa vihollisten määrän ja niiden ampumien ammusten tasapainottamista. Ensimmäisellä vihollisten läpikäynnillä tasapainotin jokaisen yksin, mutta kun aloin toteuttamaan itse pelin kulkua, huomasin, että osa vihollisista olivat hyvin hankalia, jos ne olivat yhdessä. Viholliset tarvitsivat tasapainottaa uudestaan, jotta ne ovat reilumpia yhdessä. Ja aina kun muutin sitä, mitkä viholliset syntyivät yhdessä, jouduin yleensä tasapainottamaan näitä vihollisia uudestaan. Jouduin myös vähentämään muutaman vihollisen ammusten määrää, koska ne hidastivat peliä liikaa. Pelimekaniikoiden tasapainottaminen ei ole tarkkaa tiedettä, joten sitä voi hienosäätää maailman ääriin ja silti voi löytää jotain parannettavaa, joten pitää osata miettiä, mikä on hyvä tilanne lopettaa.

5 LOPPUYHTEENVETO

Suurin osa projektille antamistani tavoitteista toteutuivat joko kokonaisuudessaan, tai ainakin osittain. Pelissä on selvä alku ja loppu. Pelaaja liikkuu ja ampuu. Viholliset liikkuvat ja ampuvat. Sain yhden tason valmiiksi ja yhden bossin. Voisin todennäköisesti tuottaa useamman tason samoilla vihollisilla, mutta tavoitteeni ei ollut tehdä isoa peliä. Tavoitteeni oli enemmän kokeiluluontoinen pelintoteutusprojekti, missä tavoitteena oli saada jotain peliä muistuttavaa kasaan. Ja siinä on onnistuttu mielestäni hyvin. Sain luotua useampia vihollisia, mitkä käyttäytyivät eri lailla toisistaan ja pääsin tutustumaan miten Phaserilla voisi toteuttaa Bullet Hell -tyylistä peliä. Sain myös bossin muistuttamaan oikeaa Bullet Hell bossia. Sain peliin luotua myös vähän omaa grafiikkaa.

Alkuperäinen tavoitteeni oli tuottaa kaksi tasoa peliin, mutta se jäi valitettavasti yhteen tasoon. Bossin suunnittelu ja toteutus olivat yllättävän haastavia ja olin käyttänyt pelin tekemiseen jo paljon aikaa. Itsessään bossin tekeminen vei noin kolme viikkoa, koska keksin joko uuden tavan tehdä jonkin bossin ominaisuuden tai vaihdoin vanhan version kokonaan uuteen. Samanlainen asia kävi vihollisten kanssa. Itse vihollisten suunnittelu ei ollut aikaa vievää, mutta sitten niiden toteutus saattoi kestää hyvin vaihtelevasti. Jotkin vihollisten ensimmäiset versiot sain kasaan parissa päivässä, mutta vaativampia kokeilujani saatoin tehdä parikin viikkoa tuloksetta. En usein löytänyt toimivaa ratkaisua, jotta vihollinen toimi halutulla tavalla, tai keksin kyseisestä vihollisesta koko ajan uusia versioita ja joudun

poistamaan vanhan version lähes kokonaan. Isoin ajan viejä oli varmaan se, kun keksin vähän väliä uuden paremman tavan tehdä jonkin uuden vihollisen osan, mikä pakotti minut palaamaan vanhoihin vihollisiin ja vaihtamaan niiden koodia tähän uuteen tapaan. Tämä tapahtui useammin kuin kerran ja se alkoi lähestymään pistettä, missä en tehnyt viikkoon muuta kuin hienosäädin vanhoja vihollisia. Samoin ammusten liikerataideat olivat yllättävän haastavia ja en pystynyt toteuttamaan läheskään kaikkia ideoitani. Halusin liikuttaa ammuksia S-mutkalla, mutta en löytänyt tapaa millä saisin järkevästi muutettua jo ammuttujen ammusten liikeratoja. Toisena esimerkkinä yritin tehdä ammuskuviota, missä vihollinen ampuu ympyrän muodossa ja ammukset liikkuisivat ulospäin ja pyörisivät vihollisen ympärillä. En ole varma olivatko ongelmat lähtöisin omasta tietämättömyydestäni vai Phaserin rajoituksista.

En ollut ennen kehittänyt peliä, joten opin paljon pelin kehityksestä. Opin miten Phaserilla on hyvä luoda objecteja. Ilmassa liikkuvien ammusten seuranta oli myös itselleni uutta, mutta onneksi siihen löytyi helppoja ohjeita ja esimerkkejä foorumeilta. Opin miten Phaserin valmiit ominaisuudet toimivat ja miten niitä kannattaa hyödyntää. Opin myös, kuinka saman asian voi tehdä useammalla eri tavalla ja mikä niistä on omalla kohdallani paras ratkaisu. Opin miten vihollisille voi tehdä simppeleitä liikeratoja. Opin miten peliä voi testata ja hioa paremmaksi tai mukavammaksi.

LÄHTEET

Arcade-history. (n.d.a). Scramble [Model GX387]. Haettu 22. 2 2019 osoitteesta https://www.arcade-history.com/?n=scramble-model-gx387&page=detail&id=2328

Arcade-history. (n.d.b). Batsugun [Model TP-030]. Haettu 27. 2 2019 osoitteesta https://www.arcade-history.com/?n=batsugun-model-tp-030&page=detail&id=193

Batsugun. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/b/b4/ARC_Batsugun.png

Bertie the Brain - Life. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/b/b4/Bertie_the_Brain_-_Life.jpg

Store.steampowered. (n.d.a).BioShock. Haettu 21. 3 2019 osoitteesta https://steamcdn-a.akamaihd.net/steam/apps/7670/0000002509.1920x1080.jpg?t=1546282147

Cambridge Dictionary. (n.d). Meaning of video game in English. Haettu 13. 2 2019 osoitteesta https://dictionary.cambridge.org/dictionary/english/video-game

Store.steampowered. (n.d.b).Cities: Skylines. Haettu 21. 3 2019 osoitteesta https://steamcdn-a.akamaihd.net/steam/apps/255710/ss_697bd257c7c02a451ce19c0542fafd3d9c25f164.1920x1080.jpg?t=1552038943

Cohen, D. (2019a). Cathode-Ray Tube Amusement Device: The First Electronic Game. Haettu 21. 3 2019 osoitteesta https://www.lifewire.com/cathode-ray-tube-amusement-device-729579

Cohen, D. (2019b). OXO aka Noughts and Crosses - The First Video Game. Haettu 18. 2 2019 osoitteesta https://www.lifewire.com/oxo-aka-noughts-and-crosses-729624

Crimzon Clover. (n.d). Wikipedia. Haettu 28. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Crimzon_Clover

Store.steampowered. (n.d.c).Crimzon Clover. Haettu 21. 3 2019 osoitteesta https://steamcdn-a.akamaihd.net/steam/apps/285440/ss_c036ed88ac8857d6b86a9f5ec59cd1971f4eacbe.1920x1080.jpg?t=1471488839

Defender. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/4/43/Defender_Gameplay_Screen.jpg

Phaser. (n.d). fantasy-wars2. Haettu 21. 3 2019 osoitteesta https://phaser.io/content/news/2018/12/fantasy-wars2.png

Galaxian. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/0/09/Galaxian.png

Godot logo. (2015). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Godot_(game_engine)#/media/File:Godot_logo.svg

Gradius. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/fi/a/a6/Gradius_04.png

Ikaruga. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/c/cf/Ikaruga_01.jpg

Krogh-Jacobsen, T. (2018). Introducing Unity 2018.3. Haettu 21. 3 2019 osoitteesta https://blogs.unity3d.com/2018/12/13/introducing-unity-2018-3/

Lambert, B. (2008). Brookhaven Honors a Pioneer Video Game. Haettu 18. 2 2019 osoitteesta https://www.nytimes.com/2008/11/09/nyregion/long-island/09videoli.html?_r=2

Nimrod (computer). (n.d). Wikipedia. Haettu 18. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Nimrod_(computer)

Spacewar! (n.d). Wikipedia. Haettu 18. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Spacewar!

Space Invaders. (n.d). Wikipedia. Haettu 21. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Space_Invaders

Galaxian. (n.d). Wikipedia. Haettu 21. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Galaxian

Ozma Wars. (n.d). FANDOM. Haettu 21. 2 2019 osoitteesta https://snk.fandom.com/wiki/Ozma_Wars

Defender (1981 video game). (n.d). Wikipedia. Haettu 22. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Defender_(1981_video_game)

Arcade-history. (n.d.c). Xevious. Haettu 25. 2 2019 osoitteesta https://www.arcade-history.com/?n=xevious&page=detail&id=3217

Gradius (video game). (n.d). Wikipedia. Haettu 25. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Gradius_(video_game)

Raiden (video game). (n.d). Wikipedia. Haettu 25. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Raiden_(video_game)

Touhou Project. (n.d). Wikipedia. Haettu 27. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Touhou_Project

Ikaruga. (n.d). Wikipedia. Haettu 28. 2 2019 osoitteesta https://en.wikipedia.org/wiki/Ikaruga

Amazon. (n.d). AWS Cloud9. Haettu 21. 3 2019 osoitteesta https://aws.amazon.com/cloud9/

Tvtropes. (n.d). Bullet Hell. Haettu 21. 3 2019 osoitteesta https://tvtropes.org/pmwiki/pmwiki.php/Main/BulletHell

Phaser. (n.d). Desktop and Mobile HTML5 game framework. Phaser. Haettu 21. 3 2019 osoitteesta https://phaser.io/

Early history of video games. (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Early_history_of_video_games

EPIC Games. (n.d). Features. Haettu 21. 3 2019 osoitteesta https://www.unrealengine.com/en-US/features

Godot (game engine). (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Godot_(game_engine)

Classicgaming. (n.d). HISTORY OF SPACE INVADERS. Haettu 21. 3 2019 osoitteesta http://www.classicgaming.cc/classics/space-invaders/history.php

History of video games. (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/History_of_video_games

Nim. (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://fi.wikipedia.org/wiki/Nim

Phaser (game framework). (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Phaser_(game_framework)

Shoot 'em up. (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Shoot_%27em_up

Godot. (n.d). THE GAME ENGINE YOU WAITED FOR. Haettu 21. 3 2019 osoitteesta https://godotengine.org/

Unity. (n.d). The world’s leading real-time creation platform. Haettu 21. 3 2019 osoitteesta https://unity3d.com/unity

Unity (game engine). (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Unity_(game_engine)

Unreal Engine. (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Unreal_Engine

Arcade-museum. (n.d.a). Xevious. Haettu 21. 3 2019 osoitteesta https://www.arcade-museum.com/game_detail.php?game_id=10505

Nimrod. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/commons/0/0c/Nimrod_in_Computerspielemuseum.jpg

Ottawa Citizen. (1975). Bertie the Brain programmer heads science council. Haettu 18. 2 2019

OXO. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/5/59/OXO_emulated_screenshot.png

Ozma Wars. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/3/31/Ozma_wars01.png

Perfect Cherry Blossom. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/8/8f/Perfect_Cherry_Blossom_screenshot.jpg

Phaser. (n.d).Phaser logo. Haettu 21. 3 2019 osoitteesta https://phaser.io/images/img.png

Pong. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/commons/thumb/b/b8/Pong_%2828684491143%29.jpg/1920px-Pong_%2828684491143%29.jpg

Punk, P. t. (2015). Defender (Arcade, 1981) - Video Game Years History. Haettu 21. 3 2019 osoitteesta https://www.youtube.com/watch?v=X-L80KM9gM8

Raiden. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/3/39/ARC_Raiden.png

Arcade-museum. (n.d.b).Scramble. Haettu 21. 3 2019 osoitteesta https://www.arcade-museum.com/images/118/1181242158132.png

Space Invaders. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/2/20/SpaceInvaders-Gameplay.gif

Imdb. (n.d).Spacewar! Haettu 21. 3 2019 osoitteesta https://www.imdb.com/title/tt0396224/mediaviewer/rm1965232640

Google play store. (n.d).Stereobreak. Haettu 21. 3 2019 osoitteesta https://lh3.googleusercontent.com/M_o2oD_foOs73UssDlxwUZAbLJC05PX9Gv9qCB4GoyWfWmuvTAERvTyYDAVSCpYo2nfC=w720-h310-rw

Tennis For Two. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/commons/b/b4/Tennis_For_Two_re-created_in_1997.png

Unity Technologies logo. (2011). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Unity_(game_engine)#/media/File:Unity_Technologies_logo.svg

Unreal Engine Logo. (n.d). Wikipedia. Haettu 21. 3 2019 osoitteesta https://en.wikipedia.org/wiki/Unreal_Engine#/media/File:UE_Logo_Black_Centered.svg

Xevious. (n.d). Wikimedia. Haettu 21. 3 2019 osoitteesta https://upload.wikimedia.org/wikipedia/en/e/e7/Xevious_Start2.png