Bullet hell-peli Unity-pelialustalle

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu, Tieto- ja viestintätekniikka

kevät, 2019

Teemu Raussi

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka

HAMK, Riihimäki

Tekijä Teemu Raussi **Vuosi** 2019

Työn nimi bullet hell-peli Unity-pelialustalle

Työn ohjaaja /t Petri Kuittinen

TIIVISTELMÄ

Opinnäytetyöni on Unityllä tehty pc-peli. Opinnäytetyöni tavoite oli opetella Unityn käyttöä 2d-alustalle ja samalla kehittyä pelikehityksessä ja c#-ohjelmointikielen käytössä. Työssä paneudutaan Unityn käyttöön, vertaillaan sitä eri pelimoottoreihin, sekä luotiin bullet hell tyylinen demopeli. Työssä käydään läpi myös bullet hell-pelien historiaa.

Peli toteutettiin c#-ohjelmointikielellä Unity-pelimoottoria hyödyntäen sekä Visual studio ohjelmakehitysympäristöä, jolla kirjoitettiin pelissä käytetty ohjelmakoodi. Pelissä käytetty grafiikka tehtiin pikselapp-selainsovelluksella.

Peli sisältää kaksi tasoa, päävalikon ja ohjeen.

Avainsanat Ohjelmointi, Pelikehitys, Unity

Sivut 30 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology

HAMK, Riihimäki

Author Teemu Raussi **Year** 2019

Subject Bullet hell game with Unity

Supervisors Petri Kuittinen

ABSTRACT

The subject of this project was video game made with Unity. The aim of the work was to learn how to use Unity in a 2D environment and to get more experience of game development and to develop with c# programming language. In this project I delve into how to use the Unity-game engine, what are differences between other game engines and create a bullet hell style game demo and go through some history of the bullet hell genre.

The game was made with the C sharp programming language using the Unity game engine and the Visual studio development environment, which was also used to write the code in the game. The graphics what was used in this game was made with a Piskellapp browser application.

The game includes two levels, main menu, and instructions on how to play.

Keywords Coding, Unity, Video-game development

Pages 30 pages

SISÄLLYS

1 JOHDANTO

Videopelit ovat olleet lapsuudesta asti minulle rakas harrastus. Ensimmäinen muisto oli Commodore 64 Nero 2000 pelin pelaaminen olemattomalla lukutaidolla. Pelien tekeminen ja suunnittelu on aina kiinnostanut ja nuoruudessani koodailin veljeni kanssa C-kielellä tekstiseikkailuja sekä kavereiden kanssa piirsimme ja suunnittelimme paperille karttoja, hahmoja ja pelasimme niillä. Siksipä opinnäytetyönkin aiheeksi valikoitui pelin suunnittelu ja tekeminen.

Tässä opinnäytetyössä oli tarkoitus tehdä Unity-pelimoottoria hyödyntäen bullet-hell tyyppinen peli pc:lle, jossa keskityttiin luomaan vihollisille tekoäly ja luodeille algoritmi erilaisille kuviolle, kuinka ammukset liikkuvat ja käyttäytyvät.

Oppinäytetyössä paneudutaan Unity-pelimoottoriin ja sen erilaisiin ominaisuuksiin, verrataan Unityä markkinoiden eri pelimoottoreihin. Käydään läpi Shoot-em'up ja sen alagenren bullet hell-pelien historiaa sekä käydään läpi pelin toteutusprosessia ja sen eri vaiheita.

2 UNITY-PELIMOTTORI

2.1 Yleistä

Unity on Unity Technologisin kehittämä pelimoottori, joka sai alkunsa 2000 luvun alussa tanskassa, kun David Helgason, Joachim Ante ja Nicholas Francis aloittivat kehittämään Apple Mac laitteille edullista pelinkehitys alustaa. Ensimmäinen versio julkaistiin vuonna 2005 ja heti perään tuki Windows-alustalle ja selaimille. Vuonna 2008 Unityn suosio kasvoi räjähdysmaisesti, kun Apple toi iPhone App Storen ja Unity oli ensimmäinen pelimoottori, joka tuki iPhoneja. Vuonna 2009 Unity sai kolme suurta asiakasta, kun Microsoft, Electronics Arts ja Ubisoft alkoivat käyttämään Unityä projekteissaan. (Brodkin, 2013)

Uusin Unity versio 2018.3 julkaistiin joulukuussa 2018 joka sisälsi monia parannuksia Prefabien käyttöön, fysiikka moottoriin, 2D-grafiikan luomiseen sekä ympäristön luomiseen tarkoitetun työkalun uudistamista ja parantelua. (Krogh-Jacobsen, 2018)

Nykyään Unity on maailman laajemmin käytetty kehitysympäristö, joka käsittää melkein kolme miljardia laitetta koko maailmassa, noin 50% kaikista mobiilipeleistä on kehitetty Unityllä. Unity technologiesissa työskentelee noin 2000 henkilöä 27 eri paikassa ympäri maapalloa ja se on maailman 7. nopeimmin kasvava työllistäjä. (Unity3Da, n.d)

Unity tukee yli 25:tä eri kehitysalustaa: (Unity3Da, n.d)

- Wii U
- Xbox One
- Oculus rift
- iOS
- Andoird
- Windows
- Steam VR
- WebGL
- PSVITA
- Apple
- Nintendo Switch
- Nintendo DS
- Playstation
- GearVR

2.1.1 Unityn asennus

Unityn asennus tapahtuu Unityn sivujen kautta, jossa valitaan haluttu lisenssi, jonka jälkeen ladataan haluttu ohjelma. Valittavana on Unity 5:n Personal-, Pro- ja Plus- lisenssit, joista Personal on ilmainen. (StoreUnity, n.d)

2.2 Lisenssit

2.2.1 Unity Personal

Unity Personal on Unityn maksuton versio, ja se on saataville yrityksille, joiden vuosittainen liikevaihto tai rahoitus on alle 100 000 dollaria. Personal versio sisältää melkein samat toiminnot kuin Pro- ja Plus- lisenssit, pois lukien muutaman pilvipalvelun ja tukipalvelun käytön. Ilmaisversiossa tehdyissä peleissä esiintyy myös Unityn-logo peliä käynnistäessä, jota ei saa vaihdettua tai otettua pois. (compare-plans, n.d)

2.2.2 Unity Pro

Unity Pro on Unityn kallein versio ja se sisältää kaikki palvelut ja rajattoman liikevaihdon tai rahoituksen. Unity Pro:n voi ostaa joko kuukausimaksuna 125 dollarin hintaan tai vuosittain tapahtumana maksuna 1500 dollarin hintaan. (compare-plans, n.d)

2.2.3 Unity Plus

Unity Plus on pienten yrityksien ja harrastelijoille tarkoitettu maksullinen versio se on saatavilla, jos vuosittainen liikevaihto tai rahoitus on alle 200 000 tuhatta dollaria. Plus-versio sisältää parannetun tuen ilmaisversioon verrattuna sekä paremman versiohallinta työkalun, sekä tarjoaa kerran kuussa ammattilaisen verkkotapaamisen. Plus-version voi ostaa 35 dollarin kuukausimaksuna tai vuosittain 300 dollarin maksuna. (compare-plans, n.d)

2.3 Unity vertailussa muihin pelimoottoreihin

2.3.1 Unreal Engine

Vuonna 1998 julkaistu Unreal-tietokonepeliä varten Epic gamesin kehittämä pelimoottori Unreal Engine on yksi suosituimmista pelimoottoreista Unityn rinnalla (deals, 2016). pelimoottorista on julkaistu 4 versiota. Näistä uusimpina Unreal Engine 4.

Molemmat pelimoottorit Unity ja Unreal tukevat 2D- ja 3D-grafiikkaa ja uusimpia graafisia renderöintitekniikoita ja valaistuksia. Kuitenkin Unreal Enginen varjot ja valot näyttävät paremmilta pienemmällä vaivalla sekä Unreal pitää sisällään monipuolisemman materiaalieditorin, jolla käyttäjä voi luoda helpommin näyttävämpiä erilaisia materiaaleja. Jos siis haluaa visuaalisesti näyttävämpiä 3D-pelejä hieman paremmilla ja monipuolisimmilla 3D työkaluilla, Unreal Engine on hieman edellä Unityä. 2D-pelien kehityksessä Unityllä on taas monipuolisemmat työkalut (Pontypants, 2018). Varsinkin mobiilipelikehityksessä ja VR sovellusten määrällä Unity on Unreal Engineä edellä, sillä yli 50% kaikista uudista mobiilipeleistä on luotu Unityllä (Unity, n.d) ja yli 60% kaikista VR ja AR sovelluksista on kehitetty Unityllä (techcrunch, 2018).

Ohjelmointikielenä Unity käyttää C#-ohjelmointikieltä, kun taas Unreal Engine C++-ohjelmointikieltä. Molemmat pelimoottorit tarjoavat valmiita paketteja yksinkertaisiin ohjelmiin ja koodipätkiin. Unityn omat Assetit ja Asset storesta löytyvät valmiit paketit helpottavat aloittelijoita, joille ohjelmointi on uutta. Unreal Enginessä on taas piirustus apuväline, joka mahdollistaa yksinkertaisen pelin tekemisen käytännössä niin, että käyttäjän ei tarvitse osata c++-kieltä ja ohjelmoida ollenkaan. Piirustus apuväline on kuin materiaalieditori esimerkiksi 3dmax sovelluksessa. Käyttäjä vetää haluttuja valmiita funkioita ja ehtoja kappaleeseen ja yhdistelee näitä laatikoita, jotta päästään haluttuun lopputulokseen (Unrealengine, n.d).

Hinnoittelussa Unreal eroaa Unitystä niin, että käyttäjä saa käyttöönsä koko ohjelman ilmaiseksi kaikilla ominaisuuksilla, mutta maksaa 5% kaikesta tuotostaan Epic gamesille (Unrealengine, n.d).
Unreal tukee yli 15:ta eri kehitysalustaa (Unrealplatforms, n.d).

2.3.2 Gamemaker: Studio

2D-pelien kehitykseen keskittynyt Gamemaker studio on vuonna 1999 YoYo gamesin kehittämä pelimoottori, joka mahdollistaa 2D-pelien tekemisen ilman suurempaa kokemusta ohjelmoinnista. Pelimoottori toimii ns. siirrä ja pudota periaatteella, jossa käyttäjä siirtelee valmiita funktioita projektiinsa ja muuttelee näiden arvoja toteuttaakseen halutun toimenpiteen.

Gamemaker:Studio on maksullinen sovellus joka tarjoaa myös kokeiluversion. Kokeiluversiossa käyttäjä voi testata sovellusta, mutta käyttö on rajattu vain rajallisiin resursseihin ja funktioihin eikä projektia voi kääntää exe-tiedostoksi. Creator lisenssin voi ostaa 39 dollarin vuosimaksulla joko Windows tai macOS käyttöjärjestelmälle. Creator lisenssissä on Unityn ilmaisversion tapaan pakotettu aloitusruutu ”splash screen”.

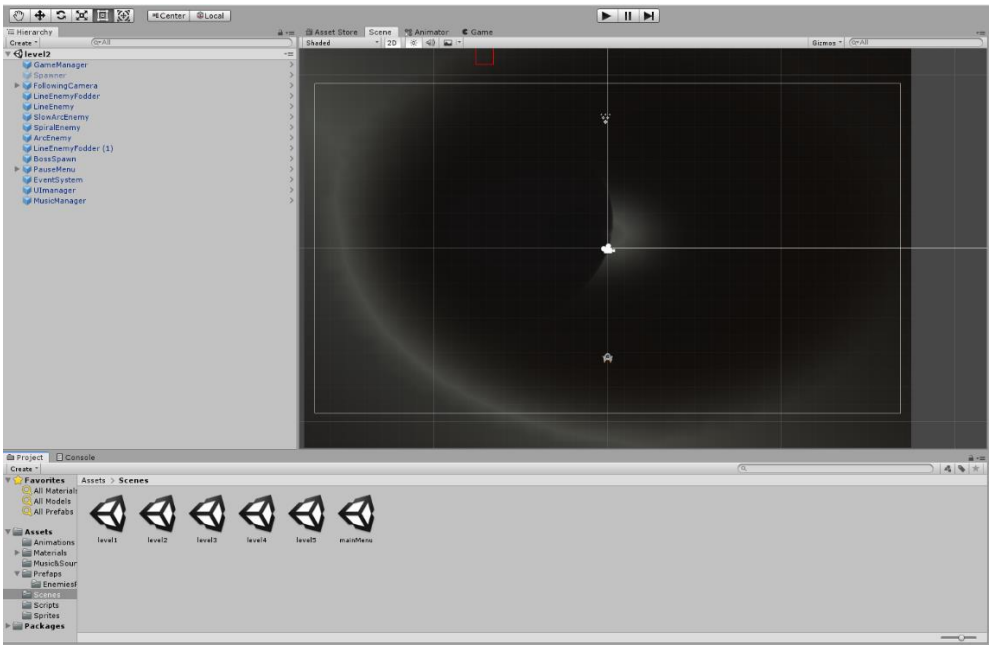
Developer lisenssi kattaa monta eri kehitysalustaa ja hinnat alkavat 99 dollarin kertamaksusta. Desktop versio mahdollistaa tehdä pelejä Windows, Mac ja Linux käyttöjärjestelmille 99 dollarin tai 83 euron hintaan Steamista. Muita kertamaksullisia lisenssejä on web-lisenssi HTML5-pelien tekemiseen 149$ tai 124,5 euron hintaan Steamista. Mobile lisenssi mobiilipelien tekemiseen 199$ tai 169.9 euron hintaan Steamista. Uwp-lisenssi Windows sekä Xbox One pelien tekemiseen 399$ tai 371.9 euron hintaan Steamista (youyougames, n.d).
GameMaker tukee yli kymmentä eri kehitysalustaa (yoyogames, n.d).

2.4 Unity käyttöliittymä

Unity tarjoaa hyvät työkalut sovellusten luontiin. Tässä kappaleessa käyn läpi Unityn perustoimintoja ja työkaluja, joita käytin tässä projektissa.

2.4.1 Scene

Scene on Unityn perusnäkymä editorissa (kuva 1), joihin lisätään peliobjektit, Ui-näkymät ja toiminnallisuus. Sovellusta aloittaessa ensimmäiseksi luodaan yleensä alkuvalikko tai testitaso, jolla koodia ja sen käytännöllisyyttä on helppo testata. Pelin lopullista kääntämistä varten otetaan halutut scenet mukaan Unityn koontiversioon järjestyksessä (kuva 2), miten halutaan pelin lataavan ensimmäisen scenen, tämä on yleensä alkuvalikko.



Kuva 1. Unityn scene näkymä editorissa

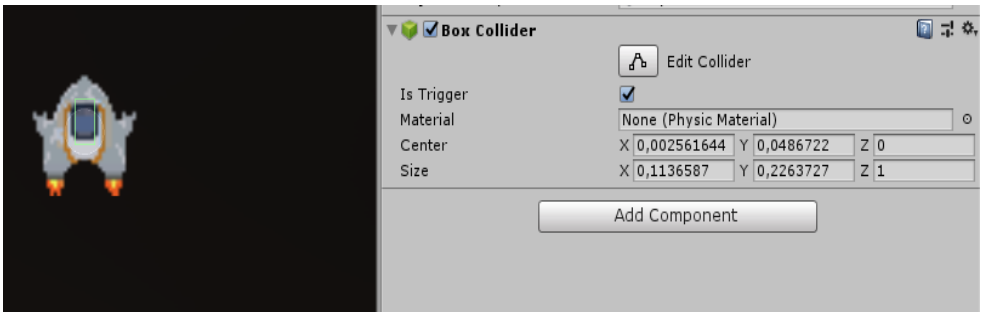
Kuva 2. Unityn koontiversio hallinta

2.4.2 GameObject

GameObject eli peliobjekti on tärkein kappale Unityssä. Peliobjekti voi olla esimerkiksi pelihahmo, ympäristöä, valoja tai kamera. GameObjectiin lisätään haluttuja komponentteja, joita ovat esimerkiksi erilaiset skriptit.

2.4.3 Collider

Collider on peliobjektiin lisätty komponentti, jonka avulla kappaleelle saadaan luotua fysiikka törmäyksiä varten esimerkiksi, kun pelaaja osuu viholliseen luotiin. Collider tarvitsee lisäksi peliobjektiin rigidbody- komponentin, jotta kappaleelle saadaan fysiikat Unityn sisällä, nämä kattavat painovoiman ja kappaleen massan. Collider- komponentteja on eri muotoisia sen mukaan kuinka tarkka törmäyksen pitää olla osuessaan peliobjektiin. Tässä pelissä käytän box Collider- komponenttia, jonka olen pienentänyt hieman pienemmäksi kuin pelihahmo. (kuva 3)



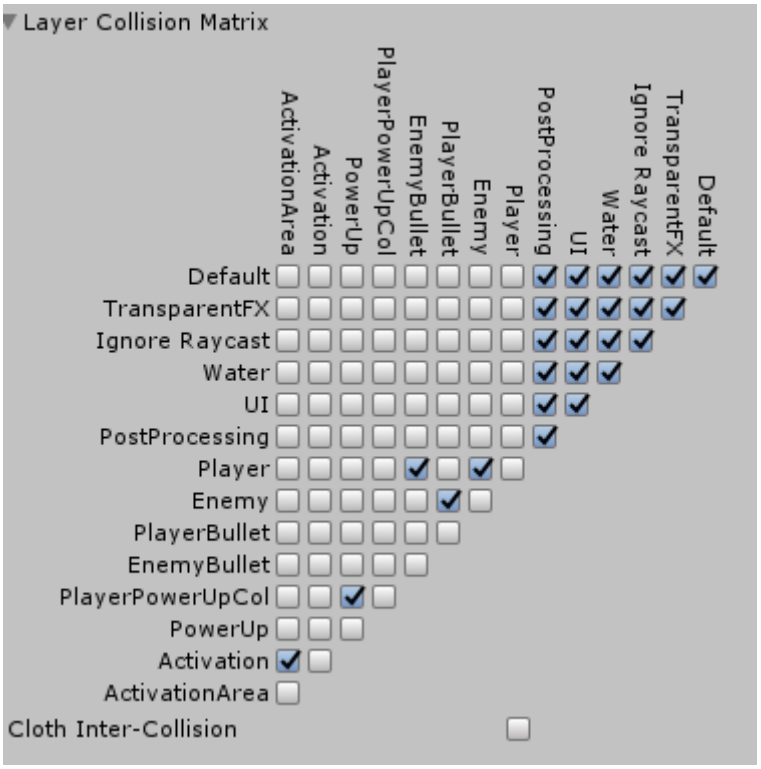
Kuva 3. Pelaajan collider-laatikko.

2.4.4 Prefab

Unityn prefab-järjestelmä mahdollistaa peliobjektien arvojen, komponenttien ja sen arvojen tallentamisen suoraan käyttäjälle talteen. Prefabeja voidaan käyttää esimerkiksi uudessa scenessä vetämällä haluttu peliobjekti sceneen ja käyttäjällä on käytössä valmis kopio alkuperäisestä peliobjektista. Tämä nopeuttaa uusien scenejen rakentamista, sillä käyttäjän ei tarvitse asettaa tai säätää peliobjektin arvoja uudestaan, kun haluaa käyttää samaa peliobjektia useasti scenessään.

2.4.5 Layers

Layerin avulla saadaan erilaisille peliobjekteille luotua tasoja, joita hallitsemalla saadaan estettyä tiettyjen collidereiden törmäystä. Esimerkiksi vihollisen luoti ei voi osua itsensä kanssa tai toiseen viholliseen, mutta osuu pelaajaan (kuva 4) tai luoda pelaajalle rajat pelitasossa, johon pelaaja ei pääse, mutta esimerkiksi pelaajaan ammukset läpäisevät tämän seinän.



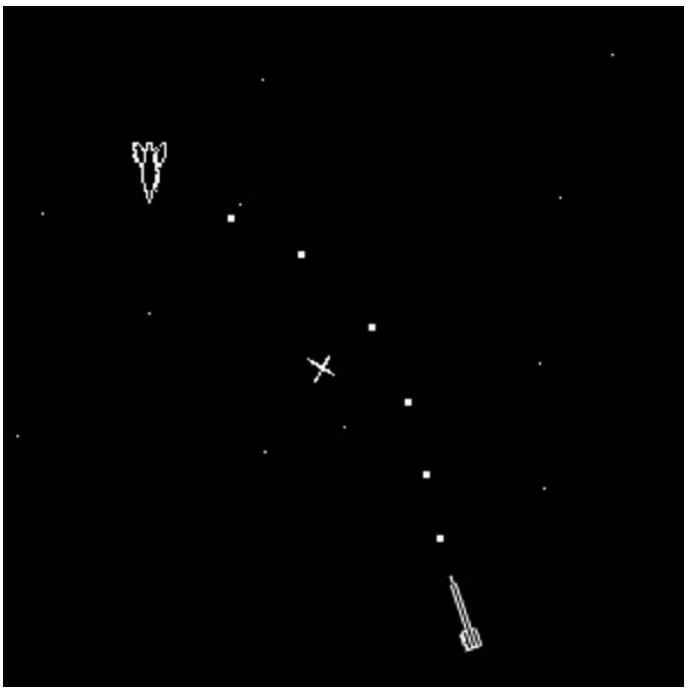
Kuva 4. Unityn layerien hallinta

3 SHOOT-EM UP GENRE

Yleisesti genren peleissä ammutaan vihollisjoukkoja tai väistellään niiden tulitusta. Joissain shoot'em up -peleissä pelaaja tuhoutuu yhdestä osumasta ja toisissa peleissä pelaajalla on käytettävinään lisäelämiä. Tyypillisesti peleissä on paljon tuhottavia vihollisia ja viholliset tulevat yleensä aaltoina tai erilaisissa muodostelmissa pelaajan kimppuun. Pelaajalle annetaan tyypillisesti tämän genren peleissä erilaisia voimanparannuksia, joko vihollisaaltoja tuhoamalla tai tason aikana kerättynä, jotka muuttavat pelaajan aseita paremmiksi, antavat lisäelämiä tai suojaavat vihollisten luodeilta hetkellisesti. (Rojas, gaminghistory101, 2012a)

3.1 Shoot'em up genren historia

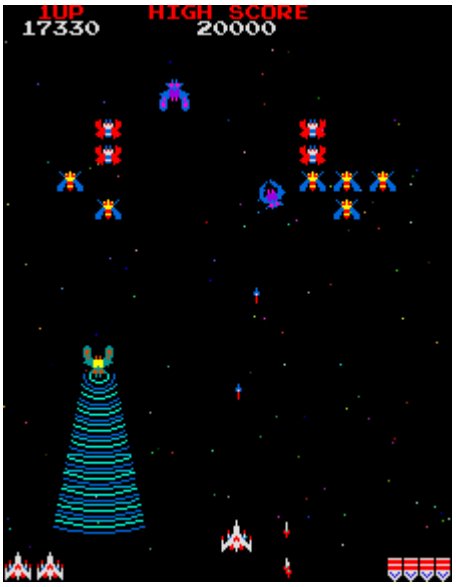
Shoot'em up genre tai paremmin tunnettuna lyhenteellä "shmup" on yksi vanhimmista videopeli genreistä ja ensimmäinen videopeli, joka voidaan luokitella shmup genreen oli Spacewar! joka esiteltiin vuonna 1962 MIT:n Science Open House tapahtumassa. (Graetz, 1981) Pelin kehittivät MIT:n opiskelijat Martin Graetz, Steve Russel ja Wayne Wiitanen PDP-1 minitietokoneelle. Spacewar! pelissä kaksi pelaajaa taistelivat toisiaan vastaan samanaikaisesti tasossa, jossa keskellä ruutua oleva auringon painovoima veti aluksia ja ammuksia kohti keskustaa. Pelaajat pyrkivät tuhota toisensa ampumalla toisiaan ja samalla liikkuivat ympäri pelitasoa (kuva 5). (Rojas, gaminghistory101, 2012a)



Kuva 5. Spacewar! (Imdb, n.d)

Vuonna 1978 genressä alkoi näkymään tyypillisen nykyaikaisen shoot-em'up genren piirteitä, kun japanilainen videopeliyritys Taito toi markkinoille videopelin Space Invaders (Giantbomb, n.d). Peli muistutti jo melko paljon nykyaikaisia shmup genren pelejä vihollisaaltoineen ja kuinka pelaajan on väisteltävä vihollisten luoteja samalla tuhoten vihollisten aluksia vaikeusasteen noustessa pelin edetessä. 1980-luvulle siirryttäessä kolikkopelien kulta-aikaan genre kehittyi tekniikan kehittyessä ja genre alkoi saamaan lopullista muotoaan.

1980-luvun alussa suurin osa genren peleistä olivat ns. yhden ruudun ammuntapelejä, jossa taso koostui liikkumattomasta ruudusta ja pelaaja pystyi vain liikkumaan yhden akselin suuntaisesti. Yksi 1980-luvun tunnetuimpia tämän tyylisiä pelejä oli Namcon vuonna 1981 julkaisema kolikkopeli Galaga (kuva 6) (Giantbomb, n.d). Galaga päivitti shmup pelien kaavaa antamalla pelaajalla mahdollisuuden ampua useamman luodin kerrallaan ja antamalla lisäpisteitä kentän läpäisystä. Vihollisilla oli myös mahdollista varastaa pelaajan alus, jolla pelaaja menetti lisäelämänsä, mutta pelaajalla oli mahdollisuus saada aluksensa takaisin tuhoamalla tämän vihollisen, joka kaappasi pelaajan. Tuhoamalla tämän pelaajalla oli mahdollisuus saada peliin mukaan toisen aluksen (Rojas, gaminghistory101, 2012c).



Kuva 6. Galaga (Wikipedia, n.d)

1980-luvun puolivälissä genressä yleistyivät horisontaalisesti tai vertikaalisesti liikkuva ruutu pelitasoissa ja näistä 1980-luvulla tunnetuimpia olivat Gradius (1985) ja R-type (1987). Nämä toivat genreen tasojen lopussa olevat vaikeat pomotaistelut ja paransivat jo toimivaa shoot-em'up kaavaa lisäämällä tehokkaampia ja näyttävämpiä poweruppeja. 1980-1990 luvulla erilaisia alagenrejä alkoi tulemaan perinteisen avaruusräiskinnän rinnalle. Näistä alagenreistä tunnetuimpia ovat bullet hell ja Run and gun alagenret (Barrish, 2013).

2000-luvulle tultaessa genren kulta-aika oli päättymässä, kun 3d pelit ja pelikonsolit toivat pelaajille suurempia, näyttävämpiä ja pidempiä pelejä. Suosittuja genren pelejä 2000-luvulla oli Gradius pelisarja (kuva 7), sen eri spinoff pelisarjat sekä Ikaruga (shmup, 2005). Shmup pelien suosiota 2010-luvulla on herättänyt bullehell alagenren nouseva suosio ja Steam-jakelualusta, jossa monet indie pelifirmat ovat lisänneet sinne omia pelejään, jotka yhdistelevät bullet hell-genreä muiden genrejen kanssa (Paprocki, 2017).

Kuva 7. Gradius V (pelaaja.fi, n.d)

3.2 Bullet hell

Termi bullet hell tai ”danmaku” on alakategoria Shoot-em’up genrelle, jossa pääpaino on vihollisten tuhoamisella ja satojen ruudulla olevien ammusten väistämiselle sekä erilaisten poweruppien keräämisellä tason aikana helpottaakseen tason läpäisyä. Yleisesti tasot koostuvat vihollisaalloista ja kentän lopussa taistellaan pääpomoa vastaan. Bullet hell tyyppiset pelit vaativat pelaajalta tarkkuuta liikkeissään ja vaikeimmissa genren peleissä pelaajan on opeteltava ulkoa vastustajan luotien liikeradat, jotta pelaajalla olisi edes pieninkään mahdollisuus selvitä tasosta.

3.2.1 Bullet hell-pelien historia

Ensimmäisiä bullet hell-peliksi luokiteltua peliä voidaan pitää vuonna 1993 ilmestynyttä Batsugunia, joka oli japanilaisen kehittäjä Toaplanin viimeiseksi jäänyt työ. Pelin teki tunnetuksi ruudun täyttävät vihollisten ammukset ja värikkäät ja räjähdykset. Batsugun oli ensimmäisiä pelejä, jotka pienensivät pelaajan hitboxia pienemmäksi kuin hahmo oli. Tätä metodia käytetään vielä nykyäänkin tämän genren tyylisissä peleissä. (Davison, 2013)

kun Toaplan meni konkurssiin suurin osa sen vanhoista työntekijöistä, perustivat uuden pelifirman nimeltä Cave vuonna 1994, joka tunnetaan monista bullet hell genren tyylisistä peleistään. (Davison, 2013)

Cave on yksi tunnetuimmista bullet hell pelien tekijöitä maailmassa ja hallitsee Guinnessin suuressa ennätyskirjassa ennätystä eniten bullet hell genren pelejä julkaisseena yrityksenä.(guinnessworldrecords, n.d). Ensimmäinen Caven peli oli DonPachi joka julkaistiin arcade-halleihin vuonna 1995 (Freeman, 2013).

DonPachi toi ruudulle entistä enemmän vihollisia ja ammuksia, joita pelaajan oli väistettävä ja uudenlaisen pisteytysjärjestelmän, jossa pelaaja sai enemmän pisteitä mitä nopeammin pelaaja tuhosi vihollisaaltoja peräkkäin. Toinen uudistus genreen oli uudelleen pelattavuus. Jos pelaaja läpäisi pelin, oli pelaajalla mahdollisuus uuteen kierrokseen, joka vaikeutti peliä niin, että viholliset ampuivat paljon enemmän ammuksia ja tuhoutuessaan räjähtivät vielä uusiksi ammuksiksi ruudulle. Cave on jatkanut bullet hell-pelien ja donpachi pelisarjan julkaisuja 2010-luvulle saakka (Rojas, gaminghistory101, 2012b).

4 PELIKONSEPTI

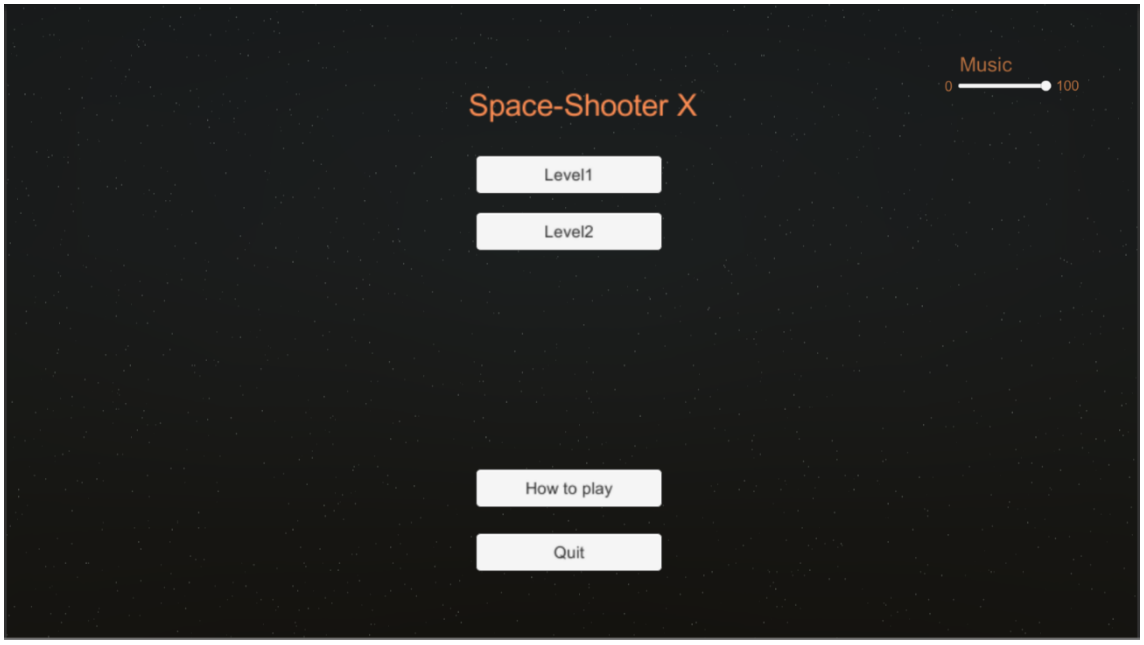
Pelin tavoitteena on väistellä ruudulla olevia ammuksia, joita on bullet hell-tyylin mukaisesti runsaasti sillä ruudulla voi olla satojakin pelaajan tuhovia ammuksia ja samalla yrittää tuhota vihollinen ampumalla tätä.

Peli on ylhäältä päin kuvattu shoot-em'up, jossa pelaaja voi liikkua ylös, alas, vasemmalle ja oikealle ruudulla, josta ylhäältäpäin tulee vihollisaaltoja ja pomo taisteluita, vihollisaaltoja tuhoamalla pelaaja saa auki parempia aseita, jotka helpottavat lopputaistelua. Pelissä on muutama erilainen taso ja pomotaistelu, joissa on erilaisia ammus kuvioita ja liikeratoja.

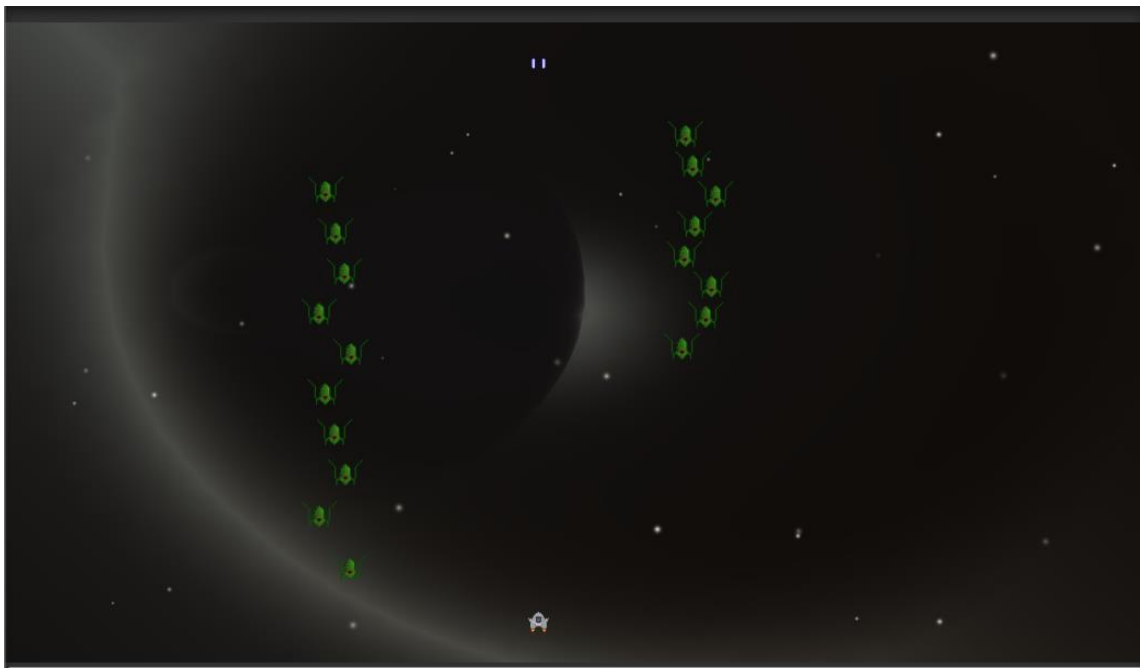
Voittaakseen vastustajan on pelaajan opittava kuinka vastustajan ammukset käyttäytyvät ja oppia väistämään ja liikkumaan niin, että pelaaja saa ammuttua sekä väistettyä vastustajaa.

5 BULLET HELL PELI UNITY-PELIMOOTTORILLA

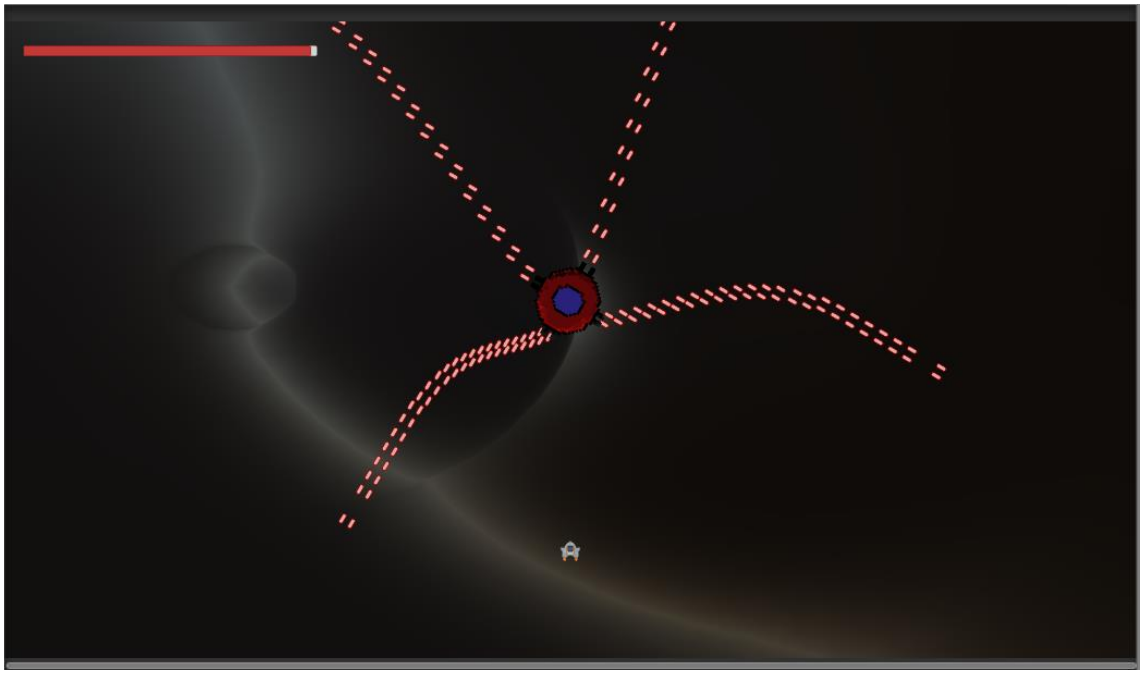
Peli toteutettiin Unity-pelimoottorin ja c#-ohjelmointikielen avulla. Scriptien ohjelmointiin käytettiin Visual Studio 2017 sovellusta. Pelissä on päävalikko, jossa näkyy pelin nimi ja josta voidaan valita kaksi erilaista tasoa ja hallita päävalikon musiikin äänenvoimakkuutta, sekä teksti tutoriaali kuinka peliä pelataan (kuva 8). Ensimmäinen taso on perinteinen shoot-em up taso, jossa vihollisaaltoja tulee ruudun yläreunasta ja pelaajan tehtävä on väistellä tai tuhota vihollisaallot (kuva 9). Kentän lopussa yläruudusta tulee pomotaistelu ja kun pelaaja tuhoaa tämän, kenttä on läpäisty. Toisessa tasossa pelaaja taistelee vain pomovastuksen kanssa ja ruudulla näkyy vihollisen elämäpalkki. (kuva 10)



Kuva 8. Pelin päävalikko



Kuva 9. Ensimmäinen taso



Kuva 10. Toinen taso

5.1 Suunnittelu

Aloitin pelin suunnitellun katsomalla millaisia erilaisia shoot-em'up pelejä on tehty ja päädyin perinteiseen arcademaiseen avaruusräiskintään. Halusin, että pelissä olisi vihollisaaltoja ja pomotaisteluita, sekä erilaisia pelaajaan hahmoa parantavia esineitä, jotka auttavat ja avaavat pelaajalle parempia tulivoimia ja aseita.

Ensimmäisen toteutin testitason, johon aloin testailemaan erilaisia vaihtoehtoja, miten pelaaja liikkuu ja millaisia vihollisia pelaaja kohtaa. Kun olin tyytyväinen perusmekaniikkaan aloin suunnitella, miten tasot toimivat ja millaisia erilaisia vihollisia on tämänkaltaisissa peleissä katsomalla tunnettuja tämän genren pelivideoita.

Sen jälkeen suunnittelin, miten toteutan vihollisen tekoälyn ja ampumisen niin, että saisin jonkinlaista kuviota pomo vihollisten ammuksista.

5.2 Pelihahmon liikkuminen ja ampuminen

Pelissä liikutaan kahden akselin suuntaisesti ylös ja alas. Liikkuminen on toteutettu yksinkertaisella scriptillä, jossa annetaan peliobjektille nopeus arvo ja liikutetaan kappaletta kahden akselin suuntaisesti, kun painetaan liikkumisnappia, jotka ovat tässä koodissa Unityn perusarvot arvoille GetAxix (vertical ja horizontal) eli nuolinäppäimet ja WASD. (kuva 11)

Pelaajan ampuminen on toteutettu scriptillä, että kun pelaaja painaa vasenta hiiren painiketta tai välilyöntiä, monistaa koodi peliobjektin ”bullet” ja ammus lähtee pelaajalle luodusta peliobjektista tulinopeudella 0,2 ruutua kohden sekunnissa (kuva 12).

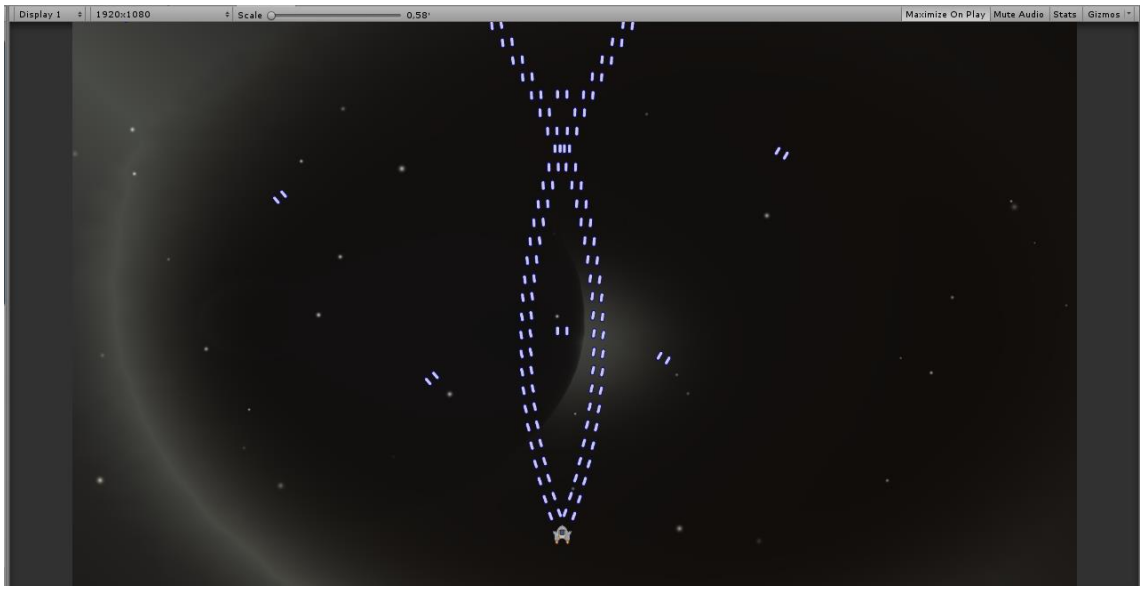
Kolme ammusta lähtee pelaajan edestä ja sivuilta ja kaksi viimeistä tykkiä, jotka pelaaja voi saada auki keräämällä poweruppeja, lisäävät uudenlaiset ammukset, jotka ampuvat kehämäistä kuviota suuremmalla tulinopeudella (kuva 13). Ampumisen skripti on lisätty komponentiksi pelaajan tykkeihin. (kuva 14)

```
public class PlayerMovement : MonoBehaviour
{
    public float speed = 1.0f;

    void Start()
    {

    }


    void Update()
    {
        transform.position += new Vector3(Input.GetAxis("Horizontal"),
            Input.GetAxis("Vertical"), 0.0f) * speed * Time.deltaTime;
    }
}
```

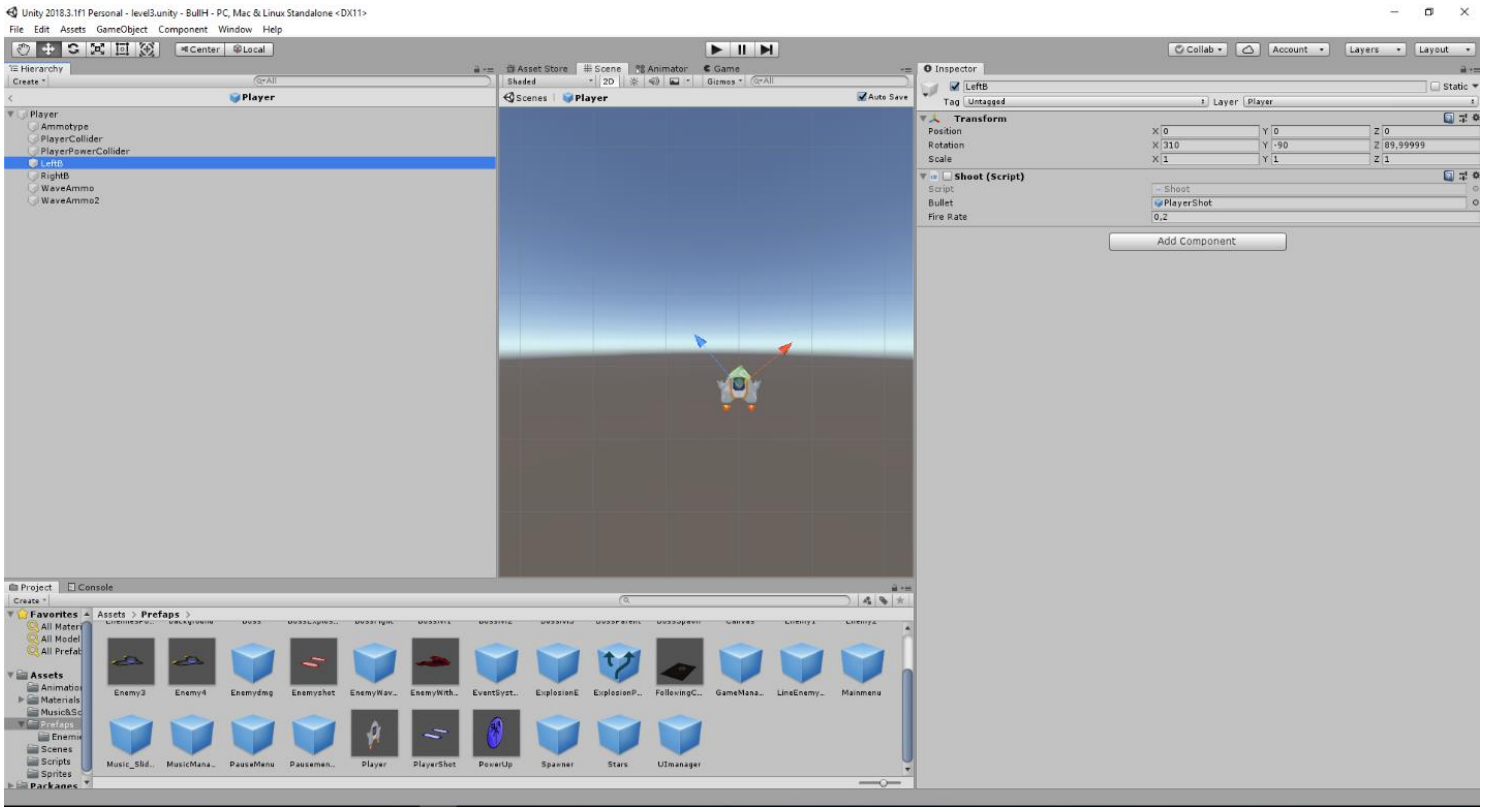
Kuva 11. Pelaajan liikkumis-skripti

```
public GameObject bullet;
public float fireRate = 0.2f;
private bool readyShot = true;

void Update()
{
    if(Input.GetButton("Fire1") && readyShot)
    {
        Instantiate(bullet, transform.position, transform.rotation);
        readyShot = false;
        Invoke("ReloadShot", fireRate);
    }
}

void ReloadShot()
{
    readyShot = true;
}
}
```

Kuva 12. Pelaajan ampuminen



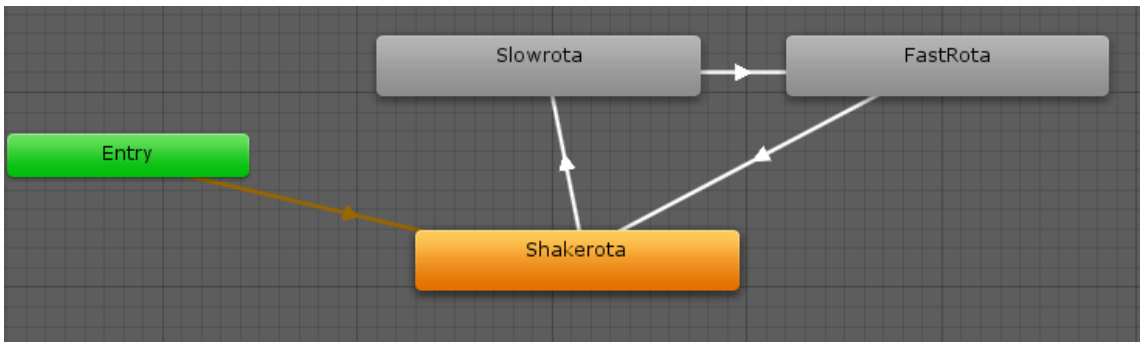
Kuva 13. Pelaaja täysissä voimissaan



Kuva 14. Pelaajan eri tykkejä ja ampumis-script asetettu pelaajan tykin komponentiksi

5.3 Vihollisen tekoäly ja liike

Pelissä on kolmenlaista erilaista vihollistyyppiä ja muutama erilainen vihollisenliikerata. Yksi vihollistyyppi on pomovastukset, jotka kestävät paljon vahinkoa ja ampuvat pelaajaa kohti monta luotia, joita pelaaja pitää varoa. Pomovastuksilla on useita eri tykkejä, joista ne ampuvat erilaisia ammuksia. Pomovastukset liikkuvat ruudulla ennalta animoituja liikeratoja pitkin, sekä kääntyilevät eri asteen kulmissa, jotta saadaan erilaisia ammuskuvioita (kuva 15).



Kuva 15. Pomovastuksen erilaisia liikeratoja animoituna

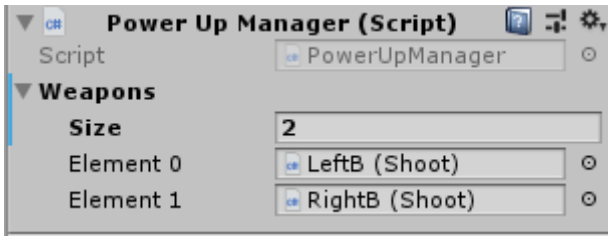
Toinen vihollistyyppi on heikot viholliset, joilla on vain yksi osumapiste ne tulevat ruudun yläreunasta aaltoina pelin edetessä erilaisilla liikeradoilla. Nämä eivät ammu pelaajaa vaan pelaajan pitää joko tuhota tai väistää viholliset. Kun pelaaja saa tuhottua koko vihollisaallon ennen kuin ne ehtivät ruudun ulkopuolelle saa pelaaja palkkioksi powerupin joka parantaa pelaajan aseita.
Kolmas vihollistyyppi on samantyylinen kuin heikot viholliset, mutta kestävät enemmän osumaa.

Vihollisaaltojen ilmestyminen on toteutettu scriptillä, joka lisätään tyhjään peliobjektiin. Skriptissä valitaan haluttu peliobjekti minkä halutaan ilmestyvän, kuinka nopeasti koodi luo monistetut peliobjektit Koodi monistaa niin monta peliobjektia kuin käyttäjä on asettanut objektiin. Skripti myös luo powerup esineen, jos pelaaja tuhoaa viimeisen vihollisen. Viimeisen vihollisen sijainnista pelialueella. (kuva 16 ja 17) Powerupit on toteutettu niin, että luodaan lista pelaajan aseista ja kun pelaaja kerää peliobjektin, joka on powerup se avaa listan mukaan sen aseen käyttöön, joka on lisätty listaan. (kuva 18 ja 19)

```
public class SpawnGroup : MonoBehaviour
{
    public GameObject objectspawn;
    public GameObject PowerUp;
    public float spawnDelay = 1.0f;
    public int numberOfEnemies;

    private int enemyCount;
    private Vector3 lastEposition;
    private bool powerUpAlive = false;

    // luo vihollis ryhmä
    void Start()
    {
        enemyCount = numberOfEnemies;
        Spawn();
    }

    private void Update()
    {
        // jos vihollisia jäljellä 0 ryhmän koosta -- luo powerup
        if (enemyCount == 0)
        {
            if (transform.childCount <= 0 && !powerUpAlive)
            {
                // kun viimeinen vihollinen tuhoutuu. luo powerup, saa viesti Dead ja lastEposition positio (spawnaa sieltä poweri)
                powerUpAlive = true;
                Instantiate(PowerUp, lastEposition, PowerUp.transform.rotation);
                Destroy(gameObject);
            }
        }
    }

    void Spawn()
    {

        enemyCount--;
        GameObject intance = Instantiate(objectspawn, transform.position, transform.rotation) as GameObject;
        intance.transform.parent = transform;
        if (enemyCount > 0)
        {
            Invoke("Spawn", spawnDelay);
        }

    }


    void Dead(Vector3 position)
        {
        lastEposition = position;
        }
```

Kuva 16. Ohjelmakoodi vihollisaaltojen ja poweruppien luomiselle

Kuva 17. Vihollisaaltojen hallinta ikkuna.



Kuva 18. Lista aseista, jotka avautuvat sen mukaan, kun pelaaja kerää powerupin.

```
{
    public Shoot[] weapons;
    private int powerLevel = -1;

    void PowerUp()
    {
        powerLevel++;
        if(powerLevel < weapons.Length)
        {
            weapons[powerLevel].enabled = true;
        }
    }
}
```

Kuva 19. Jokainen kerätty powerup lisää yhden voimatason arsenaaliin.

5.4 Tasosuunnittelu ja vaikeusaste

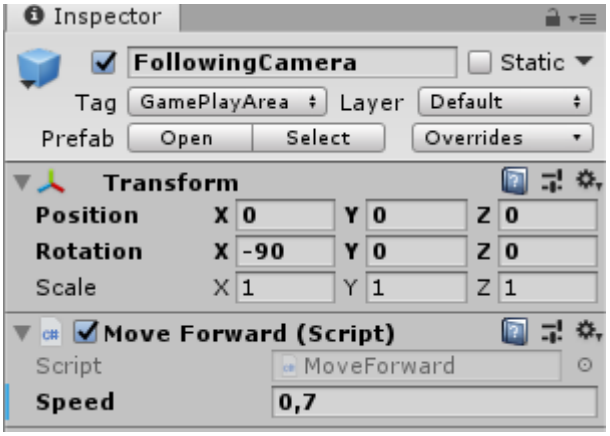
Pelissä on kaksi erilaista tasoa. Molemmissa tasoissa pelaajalla on käytettävinä vain yksi elämä ja yksi osumapiste. Pelaajan tuhoutuessa pelaaja voi aloittaa kentän uudelleen painamalla Enteriä tai Esciä avatakseen taukovalikon, josta pelaaja pääsee takaisin alkuvalikkoon. Ensimmäisessä tasossa pelaajan kimppuun tulee vihollisaaltoja koko tason ajan, kunnes aivan kentän lopussa ilmestyy kovempi ja kestävämpi vihollinen, jonka pelaajan on voitettava läpäistäkseen tason.

Toisessa tasossa pelaajaa odottaa pomovastus, jolla on elinvoimapalkki. Pomotaistelun aikana vihollisia ilmestyy ruudun yläpuolelta, joita tuhoamalla pelaaja saa poweruppeja. Pelaaja läpäisee tason, kun pomovastuksen elämäpisteet menevät nollaan. Pomovastuksen koodissa on mahdollista hallita kuinka paljon vastustaja kestää vahinkoa sekä millainen efekti syntyy, kun pelaaja osuu tai tuhoaa peliobjektin. Osuma ja tuhoutumisefekti on tehty niin, että se ilmestyy aina siitä missä peliobjekti on eikä se voi kääntyä eri kulmissa. (kuva 20)

```
public class BossHP : MonoBehaviour
{

    public int maxhealth = 10;
    private int currenthealth;
    public GameObject hitEffect;
    public GameObject deathEffect;
    public Slider healthSlider;


    private void Awake()
    {
        currenthealth = maxhealth;
    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnTriggerEnter(Collider col)
    {
        currenthealth--;
        healthSlider.value = currenthealth;
        Instantiate(hitEffect, col.transform.position, Quaternion.identity);
        if(currenthealth <= 0)
        {
            transform.parent.SendMessage("Dead", transform.position, SendMessageOptions.DontRequireReceiver);
            Instantiate(deathEffect, col.transform.position, Quaternion.identity);
            Destroy(gameObject);

        }
    }
}
```

Kuva 20. Pomoviholisen elämäpisteet UI: näytölle ja erilaiset efektit osuessaan tai tuhoutuessaan

Jokainen taso on luotu niin, että koko pelialue liikkuu hieman eteenpäin (kuva 21) ja kun kamera törmää peliobjektiin tason ulkoreunalla se suorittaa koodin, joka taas luo vihollisaaltoja pelaajan tuhottavaksi (kuva 22). Pomovastukset pidetään pelialueen sisällä scriptillä, joka perii kentän ominaisuudet ja näin estää pomovastuksen liikkumisen tason ulkopuolelle (kuva 23).



Kuva 21. Pelialueen liikkuminen

```
public class InitiateScriptsOnTrigger : MonoBehaviour
{
    void OnTriggerEnter()
    {
        foreach(MonoBehaviour mono in gameObject.GetComponentsInChildren<MonoBehaviour>())
        {
            mono.enabled = true;
        }
    }
}
```

Kuva 22. Kun pelialue osuu ulkopuolella olevaan peliobjektiin, jossa on collider- komponentti se suorittaa tämän objektin skriptin.

```
public class SetParent : MonoBehaviour
{

    public Transform target;
    public string ParentTag;


    void SetNewParent()
    {
        Transform newParent = GameObject.FindGameObjectWithTag(ParentTag).transform;

        if (target == null)
        {
            transform.parent = newParent;
        }
            else
        {
            target.parent = newParent;
        }
    }
}
```

Kuva 23. Asettaa peliobjektin halutun tagin lapseksi

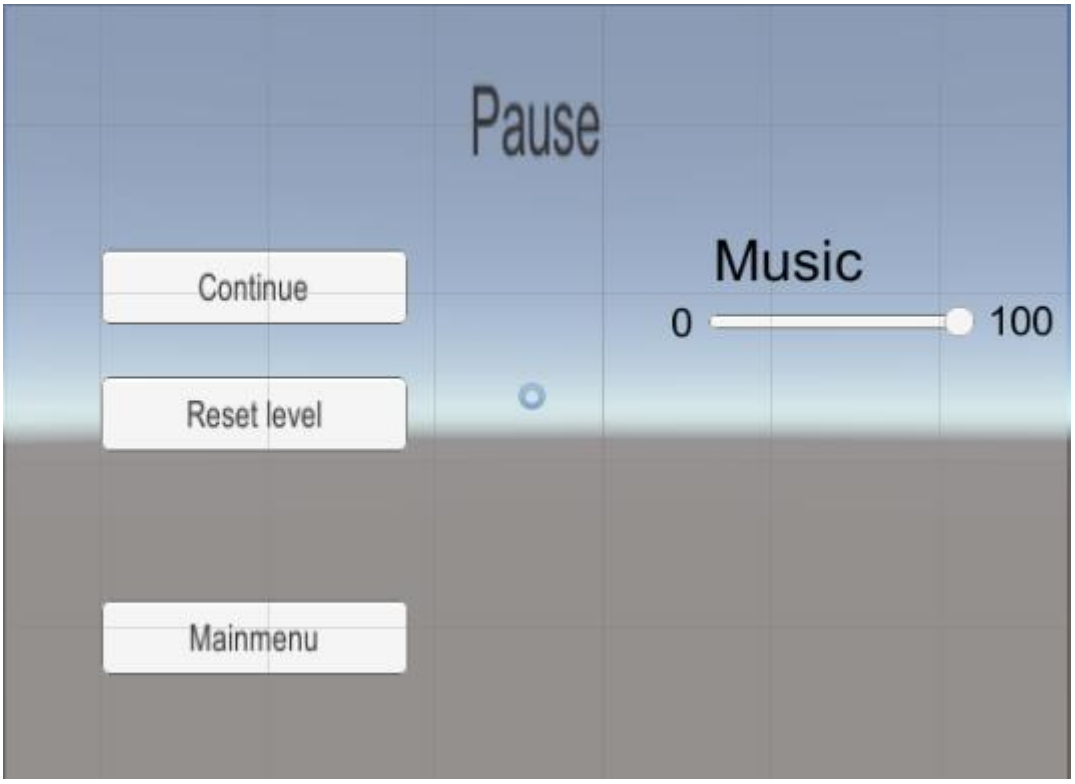
5.5 Grafiikka

Pelin grafiikan tein piskellapp-sivun pikselieditorilla. Sivustolla voi tehdä erilaisia 2D-tietokone grafiikoita suoraan selaimessa ja siirtää valmis kuva suoraan tietokoneelle käytettäväksi Unityyn (Kuva 24).

Kuva 24. Grafiikkaa, jotka on tehty piskellapp-sivulla. (piskelapp, n.d)

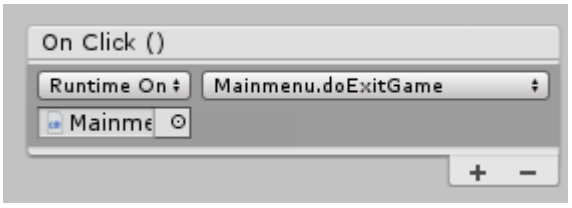
5.5.1 Käyttöliittymä

Pelissä on taukovalikko, joka aukeaa esciä painamalla. Valikossa pelaaja voi säätää musiikin äänenvoimakkuutta, jatkaa peliä, palata takaisin alkuvalikkoon tai aloittaa tason uudelleen (kuva 25).

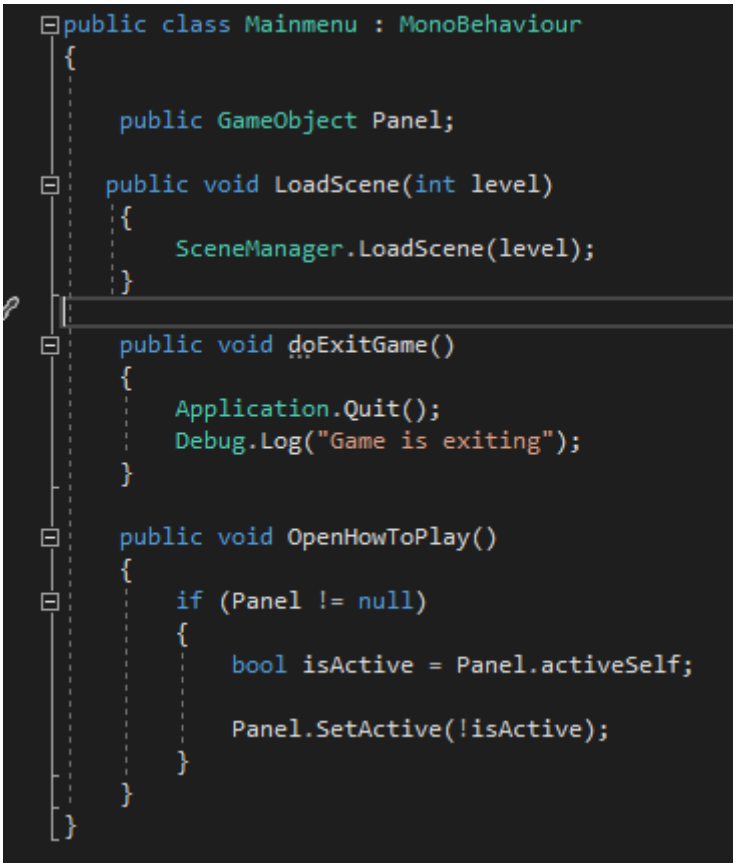


Kuva 25. Pelin pausevalikko

Aloitusvalikosta voi valita halutun tason, jota pelaaja haluaa pelata, katsoa ohjeet, miten pelataan tai poistua pelistä. Aloitusvalikko rakentuu nappuloista, joissa on OnClick funktio ja jokaiselle napille on asetettu oma funktionsa. Esimerkiksi Quit game -nappia painamalla peli suorittaa sovelluksen sulkemis- funktion. (kuva 26 ja 27)



Kuva 26. OnClick funktio Quit game -napissa.



```
public class Mainmenu : MonoBehaviour
{

    public GameObject Panel;

    public void LoadScene(int level)
    {
        SceneManager.LoadScene(level);
    }

    public void doExitGame()
    {
        Application.Quit();
        Debug.Log("Game is exiting");
    }

    public void OpenHowToPlay()
    {
        if (Panel != null)
        {
            bool isActive = Panel.activeSelf;

            Panel.SetActive(!isActive);
        }
    }
}
```

Kuva 27. Mainmenu scripti joka on asetettu Main menu peliobjektiin.

5.5.2 Efektit

Peliin on tehty kolme erilaista efektiä. Efektit ovat tehty Unityn partikkeli efekteistä. Kaksi efektiä on, kun pelaaja tai vihollinen tuhoutuu, syntyy pieni räjähdys. Kolmantena efektinä on osumatehoste pomotaisteluja varten niin, että pelaaja näkee, kun saa osuman pomoviholliseen.

5.5.3 Optimointi

Pelin monet kymmenet luodit tuhotaan, kun ne poistuvat kameran näkökentästä Unityn omalla funktiolla OnBecameInVisible (kuva 28). Tämä on tehokas ja toimiva tapa peliin, joka on graafisesti yksinkertainen, mutta ei välttämättä parhain tapa optimoida peliä, sillä se poistaa objektin vain silloin kun pelimoottori ei piirrä sitä ollenkaan. Etenkin jos peliobjekteilla olisi varjot ne voivat piirtyä vielä kauan, vaikka objekti olisi poistunut kamerasta.

```
public class Destroyoffscreen : MonoBehaviour
{
    public GameObject destroyObject = null;

    private void OnBecameInvisible()
    {
        if (destroyObject == null)
        {
            Destroy(gameObject);
        }
        else
        {
            Destroy(destroyObject);
        }
    }

}
```

Kuva 28. Tuhoa peliobjekti, kun se poistuu kamerasta

Muita tapoja optimoida luotien määrää pelissä olisi luoda kentän ympärille laatikko, joka tuhoaa ammukset välittömästi siihen törmätessään. Näin pelin ei tarvitsisi piirtää objektia kovin kauan ja määrä pysyisi melko pienenä koko pelin ajan.
Yksi parhaista tavoista optimoida tämän tyylisiä pelejä, joissa on useita luoteja kerrallaan, olisi object booling eli objektivaranto (Crook, 2014). Vaikka Instantiate ja Destroy metodit eivät olekaan kovin raskaita prosessorille niin monen sadan objektin luonti ja jatkuva tuhoaminen saattaa hidastaa vanhempia ja tehottomia prosessoreita, sillä Unity käyttää resursseja vapauttaakseen muistia ”roskankeräys” prosessissa, kun objekti tuhoutuu (Unity3Db, n.d). Object booling tapa käyttää hyödyksi jo olemassa olevia peliobjekteja, jotka ladataan jo tason tai scenen alussa, joten sen ei tarvitse luoda koko ajan lisää tai tuhota niitä.

6 YHTEENVETO JA JATKOKEHITYS

Tavoitteena oli saada aikaan yksinkertainen shoot-em'up tyylinen bullet hell-peli ja paneutua tarkemmin Unityn ja c# toiminnallisuuteen ja käyttöön. Työ kehitti hyvin Unityn käyttötaitoja ja varsinkin animoinnista ja 2D asioista oppi todella paljon uutta, sillä ne olivat minulle aivan uusia tuttavuuksia. Työstä oppi myös hyvin paljon C#-kielen erialisia ratkaisutapoja ja huomasinkin työtä tehdessä, että jokaisessa asiassa oli monta tapaa ratkaista eteen tulleet ongelmat.

Lopputulokseen olen melko tyytyväinen. Grafiikat ja efektit olisivat voineet olla hieman kauniimpia, mutta se olisi vaatinut itseltäni jonkinlaista taiteellista lahjakkuutta. Perustoiminnallisuus kuitenkin toimii ja peliä voi tuntea pelaavansa.

Jatkokehitystä ajatellen peliin olisi saada jonkinlainen oikea tason lopetusilmoitus ja erilaisia vihollisia ja pomotaisteluita. Highscore ja jonkinlainen taso tasolta eteneminen tallennuksen kanssa olisi myös oiva lisä. Myös jonkinlainen pelaajan muokkaus esimerkiksi erilaisten aloitusaseiden ja aluksen ulkonäön hallinta, sekä pelaajan tasokehitys ja kokemuspisteiden kerääminen olisi mukavaa lisätä peliin.

LÄHTEET

Barrish, C. J. (2013). *libertygames*. Noudettu 23.4.2019 osoitteesta https://www.libertygames.co.uk/blog/a-detailed-history-of-shoot-em-up-arcade-games/

Brodkin, J. (2013). *Dice*. Noudettu 23.4.2019 osoitteesta https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

compare-plans. (n.d). *Unity Store*. Noudettu 23.4.2019 osoitteesta https://store.unity.com/compare-plans

Crook, D. (2014). *blogs.msdn*. Noudettu 23.4.2019 osoitteesta https://blogs.msdn.microsoft.com/dave_crooks_dev_blog/2014/07/21/object-pooling-for-unity3d/

Davison, P. (2013). *usgamer*. Noudettu 23.4.2019 osoitteesta https://www.usgamer.net/articles/curtains-for-you-the-history-of-bullet-hell

deals, T. (2016). *thenextweb*. Noudettu 23.4.2019 osoitteesta https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/

Freeman, W. (2013). *Eurogamer*. Noudettu 23.4.2019 osoitteesta https://www.eurogamer.net/articles/2013-12-06-dodonpachi-retrospective

Giantbomb. (n.d). *giantbomb*. Noudettu 23.4.2019 osoitteesta galaga: https://www.giantbomb.com/galaga/3030-15784/

Giantbomb. (n.d). *Giantbomb*. Noudettu 23.4.2019 osoitteesta https://www.giantbomb.com/space-invaders/3030-5099/releases/

Graetz, J. M. (1981). *wheels.org*. Noudettu 23.4.2019 osoitteesta http://www.wheels.org/spacewar/creative/SpacewarOrigin.html

guinnessworldrecords. (n.d). *guinessworldrecords*. Noudettu 23.4.2019 osoitteesta http://www.guinnessworldrecords.com/world-records/most-prolific-developer-of-danmaku-shooters/#

Imdb. (n.d). Noudettu 23.4.2019 osoitteesta Imdb: https://www.imdb.com/title/tt0396224/mediaviewer/rm1965232640

Krogh-Jacobsen, T. (2018). *Blogs.Unity3d*. Noudettu 23.4.2019 osoitteesta https://blogs.unity3d.com/2018/12/13/introducing-unity-2018-3/?_ga=2.240914819.1615270833.1548680860-132076693.1547489471

Paprocki, M. (2017). *Pcgamer*. Noudettu 23.4.2019 osoitteesta https://www.pcgamer.com/how-steam-brought-shmups-out-of-arcades-and-into-a-new-pc-renaissance/

pelaaja.fi. (n.d). Noudettu 23.4.2019 osoitteesta pelaaja: https://pelaaja.fi/sites/default/files/kuvat_teksti/gradius1.jpg

piskelapp. (n.d). Noudettu 23.4.2019 osoitteesta https://www.piskelapp.com/

Pontypants. (2018). *Sunday Sundae*. Noudettu 23.4.2019 osoitteesta https://sundaysundae.co/unity-vs-unreal/

Rojas, F. (2012a). *gaminghistory101*. Noudettu 23.4.2019 osoitteesta https://gaminghistory101.com/2012/02/29/shmup/

Rojas, F. (2012b). *gaminghistory101*. Noudettu 23.4.2019 osoitteesta https://gaminghistory101.com/2012/03/12/dodonpachi/

Rojas, F. (2012c). *gaminghistory101*. Noudettu 23.4.2019 osoitteesta galaga: https://gaminghistory101.com/2012/03/22/galaga/

shmup, h. o. (2005). *nullpointer*. Noudettu 23.4.2019 osoitteesta http://www.nullpointer.co.uk/content/endless-fire-a-history-of-the-shmup/

StoreUnity. (n.d). *Unity store*. Noudettu 23.4.2019 osoitteesta https://store.unity.com/

techcrunch. (2018). *techcrunch*. Noudettu 23.4.2019 osoitteesta https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=BKHIn5i7ySLzdb-GUEfROw

ultimatehistoryvideogames. (n.d). Noudettu 23.4.2019 osoitteesta https://ultimatehistoryvideogames.jimdo.com/space-invaders/?cmsEdit=1

Unity. (n.d). *Unity mobile*. Noudettu 23.4.2019 osoitteesta https://unity.com/solutions/mobile

Unity3Da. (n.d). *public-relations*. Noudettu 23.4.2019 osoitteesta https://unity3d.com/public-relations

Unity3Db. (n.d). *optimizing-garbage-collection-unity-games*. Noudettu 23.4.2019 osoitteesta https://unity3d.com/learn/tutorials/topics/performance-optimization/optimizing-garbage-collection-unity-games

Unrealengine. (n.d). *docs.unrealengine*. Noudettu 23.4.2019 osoitteesta https://docs.unrealengine.com/en-us/Engine/Blueprints/GettingStarted

Unrealengine. (n.d). *FAQ*. Noudettu 23.4.2019 osoitteesta https://www.unrealengine.com/en-US/faq

unrealengine. (ei pvm). *UnrealEngine*. Noudettu 23.4.2019 osoitteesta https://docs.unrealengine.com/en-us/Engine/Blueprints/GettingStarted

Unrealplatforms. (n.d). *Platforms*. Noudettu 23.4.2019 osoitteesta https://docs.unrealengine.com/en-us/Platforms

Wikipedia. (n.d). Noudettu 23.4.2019 osoitteesta Wikipedia: https://upload.wikimedia.org/wikipedia/en/2/2a/Galaga.png

youyougames. (n.d). *get*. Noudettu 23.4.2019 osoitteesta https://www.yoyogames.com/get

yoyogames. (n.d). *youyougames*. Noudettu 23.4.2019 osoitteesta gamemaker: https://www.yoyogames.com/gamemaker