TAMPEREEN
AMMATTIKORKEAKOULU

THE BLACKDROP PELIN OHJELMOINTI

Miikael Lehtimäki

Opinnäytetyö
Elokuu 2017
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

LEHTIMÄKI MIIKAEL:
The Blackdrop pelin ohjelmointi

Opinnäytetyö 38 sivua
Elokuu 2017

Tämä opinnäytetyö käsittelee pelisuunnittelua ja ohjelmointia. Työni on suppea pelisuunnitelma, josta selviää pelin perusidea, pelin ohjelmointitapa ja näihin liittyvä lyhyt kuvaus pelin keskeisistä ominaisuuksista. Opinnäytetyössäni keskityin pelin ohjelmointiin ja opettelin samalla uuden pelientekoon käytettävän ohjelmiston käytön. Työ toimii apuna oman peli-idean esittelemisessä ja sen konseptin jatkotyöstössä.

Tavoitteena oli opetella uusi pelien koodaukseen soveltuvan ohjelmointi alustan käyttö ja tehdä sillä lyhyt pelisovellutus.

Kielenä tässä työssä käytin C Sharp (C# versio 7.0) ohjelmointikieltä, jota pelien kehittämiseen tarkoitettu unity- ohjelmointiympäristö tukee. Työhön liittyvät kuvat piirrettiin verkkoselaimessa toimivalla Piskel ja Windows Paint ohjelmalla, ja pelin lopputausta on kuvakaapattu animaatiosta. Pelin käyttämät äänet äänitin Windows 10:n mikrofoniohjelmalla ja editoinut ne Twisted Wave Online verkkoselain editorilla, osan äänistä otin freesound.org sivustolta ja niiden alkuperäiset tekijät on asianmukaisesti listattu lähdeluettelossa. Ohjelmoinnin tein omalla Windows 10 pohjaisella tietokoneellani tai käyttämällä sitä TeamViewer etäyhteys ohjelman kautta.

Blackdrop pelin keskeiset ominaisuudet ohjelmoin yllä kuvatulla menetelmällä neljässä eri ohjelmointikehitysvaiheessa ja kuhunkin niihin liittyvät käskysarjat ja niiden liittyminen toisiinsa pelin ohjaamisessa on kuvattu työn kuvissa yksityiskohtaisesti. Pelissä pelihahmo tippuu alaspäin ja pelaajan tehtävä on ohjata tätä hahmoa väistelemällä erilaisia esteitä ja vihollisia pelin loppuun asti. Mikäli pelaajahahmon pudotessa sen ns. elämäpisteet vähenevät vaarojen kohtaamisen myötä nollaan, se palautuu pelikentän alkuun ja jatkuu siitä eteenpäin uudestaan kohti maalia. Peliä pelataan näppäimistön ja hiiren avulla.

Työn pohjalta voidaan samalla ohjelmointimenetelmällä jatko kehittää esim. kännyköihin tai muihin pelilaitteisin soveltuva Blackdrop peli tai vastaava lyhyt sovellus. Työ toimii apuna oman peli-idean prototyypin esittelemisessä ja konseptin jatkotyöstössä.

Asiasanat: unity, c#, asset

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Technology

LEHTIMÄKI MIIKAEL:
Programming of The Blackdrop Game

Bachelor's thesis 38 pages
August 2017

This thesis handles game design and programming. My work is a slim game plan that defines the basic idea of the game, the way it is programmed and a short description of the key features of the game. My efforts were concentrated on programming the game and learning to use the new program for game design. This work serves as a tool in presenting my game idea and its concepts.

The objective was to learn how to use a new programming platform and making a small game.

The programming language chose for the work was C Sharp (C# version 7.0), which is supported by the unity environment. Images used in work were made using Piskel-browser app and Windows Paint programs, with the final background being a screenshot from an animation. Some game sounds were recorded using the Windows 10 default microphone application and edited using Twisted Wave Online browser editor, others were taken from freesound.org, with the authors adequately credited in list of sources. Programming was done on my Windows 10 desktop at home or using the TeamViewer remote connection program.

The central components of the game were programmed as described above in four different parts and the important parts of the scripts used and their interconnections are shown in pictures in detail. In the Blackdrop game its characters fall down (drop) and the gamers' task is to guide these game characters by avoiding different kind of obstacles and enemies until the end of the game. If the characters so called life point falls zero when they meets obstacles and enemies during fall, the game automatically returns to levels starting point by continues again towards target. The game is played by using a keyboard and a mouse.

From this work game development can be continued to create versions for mobile and other platforms. This works serves as a tool for presenting a prototype of the game idea and for further work on the concept.

Key words: unity, c#, asset

SISÄLLYS

ERITYISSANASTO JA LYHENTEET

asset	unityn nimi peliohjelmistokomponenteille
boo	boo tietokoneen ohjelmointikieli
bool	muuttujatyyppi joka on joko tosi tai epätosi
C#	C Sharp, tietokoneen ohjelmointikieli
Gameobject	peliesine
google	World Wide Web hakukone
iteraatio	ohjelmoinnin kehitysjakson toistaminen
Javascript	tietokoneen komentosarjakieli
käyttöjärjestelmä	keskeinen tietokoneen ohjelmisto esim. Microsoft Windows
NET rajapinna	tietokoneen ja internetin rajapinnan käyttöä helpottava ohjelma
skripti	komentosarja
tasohyppelypeli	pelit joissa pelihahmoa ohjataan hyppäämällä erilaisten tasojen päälle.

1 JOHDANTO

Tämä opinnäytetyö käsittelee pelisuunnittelua ja pelinohjelmointia. Työni on suppea pelisuunnitelma. Siitä käy selväksi pelin perusidea, pelin ohjelmointitapa, ja sen eri ohjelmointivaiheiden ja ohjelmoinnin kehitysjaksojen toistojen (iteraatioiden) koodien yksityiskohtainen sisältö, niiden toimintojen ja ominaisuuksien kuvaus. Olen myös piirtänyt koko pelin rakennetta kuvaavan kokonaisprosessikaavion, jossa esitän miten pelin eri käskysarjat toiminnallisesti kokonaisuutena liittyvät toisiinsa. Opinnäytetyössäni keskityin pelin ohjelmointiin ja opettelin minulle uuden pelientekoon käytettävän ohjelmiston käytön. Työ toimii apuna oman peli-idean esittelemisessä ja konseptin jatkotyöstössä ja kehittämisessä.

Tasohyppelypelit ovat videopelien lajityyppi, joissa pelihahmoa ohjataan hyppäämällä erilaisten tasojen päälle. ”BLACKDROP” eli musta pudotus on yksinkertainen kaksiulotteinen tasohyppelypeli, jonka koodasin tässä opinnäytetyössäni Unity-ohjelmointiympäristössä käyttäen Unityn komentosarjoja (ns. Scriptiä). Se on Unity Technologiesin kehittämä monialustainen pelikehitykseen tarkoitettu ohjelmistoympäristö, jolla voidaan kehittää kaksi- ja kolmiulotteisia selain-, konsoli- ja tietokonepelejä suosituimpiin pelikonsoleihin tai pelejä melkein mille tahansa järjestelmälle (Unity3d. 2017).

Valitsin tämän ohjelmointityökalun, koska monet pelinkehitysyhtiöt käyttävät sitä yhtenä standardi ohjelmointityökaluna. Tämän työkalun käytön etuna on se, että ohjelmistoa voi käyttää ilman että siitä täytyy maksaa erikseen ja sille löytyy selkeät käyttöohjeet ja esimerkit internetistä, sekä joukko ilmaisia sovelluksia (asset store) jotka helpottavat pelin tekemistä, työkalun itsenäistä käyttöön ottoa ja opettelua. (https://unity3d.com/unity). Lisäksi unity tarjoaa mahdollisuuden mm. 3D grafiikan ja pelien kehittämiseen.

Tässä opinnäytetyössäni valmistin tällä ohjelmalla kaksiulotteisen tasohyppelypelin, jossa pääteemana on asteittainen tasokohtainen putoaminen (pimeyteen) väistellen erilaisia pelissä eteen tulevia vaaratilanteita ja esteitä pelin loppuun asti. Peli on valmistettu (koodattu) useamman ohjelmointijakson ylitse, joille jokaiselle olen

suunnitellut pääsuuntauksen ja, joista jokainen toteutuu kunkin menossa olevan jakson aikana. Peliä pelataan näppäimistön ja hiiren avulla.

Aloittaessani tämän opetusnäytetyön minulla ei ollut aikaisempaa kokemusta unityn käytöstä. Ennen kuin aloitin, pelinkoodaustehtävän tekemisen tutustuin netistä saatavissa oleviin tietolähteisiin oppiakseni käyttämään pelien valmistukseen soveltuvaa Unity – ohjelmointiympäristöä. Apuvälineitä ohjelmointiin ja työn tekemiseen hain Googlen hakukoneella internetistä ja löysin sen avulla koodaustehtävän suorittamiseen tarvittavat ohjelmointiohjeet ja kirjallisuuskatsauksen kirjallisuusviitteet. Esimerkiksi hyvä tietopaketti Unity – ohjelmointi ympäristöstä löytyy verkkosivustolta osoitteesta (https://unity3d.com/unity). Kielenä tässä työssä käytin C Sharp (C# versio 7.0) ohjelmointikieltä, jota unity- ohjelmointiympäristö tukee ja, jota on jo aikaisemmin myös insinöörikoulutuksessamme lyhyesti sivuttu ks. (https://www.infoq.com/minibooks/emag-c-sharp-preview).

Kiitän opinnäytetyön ohjauksesta Tampereen ammattikorkeakoulun lehtoria Tony Torppia.

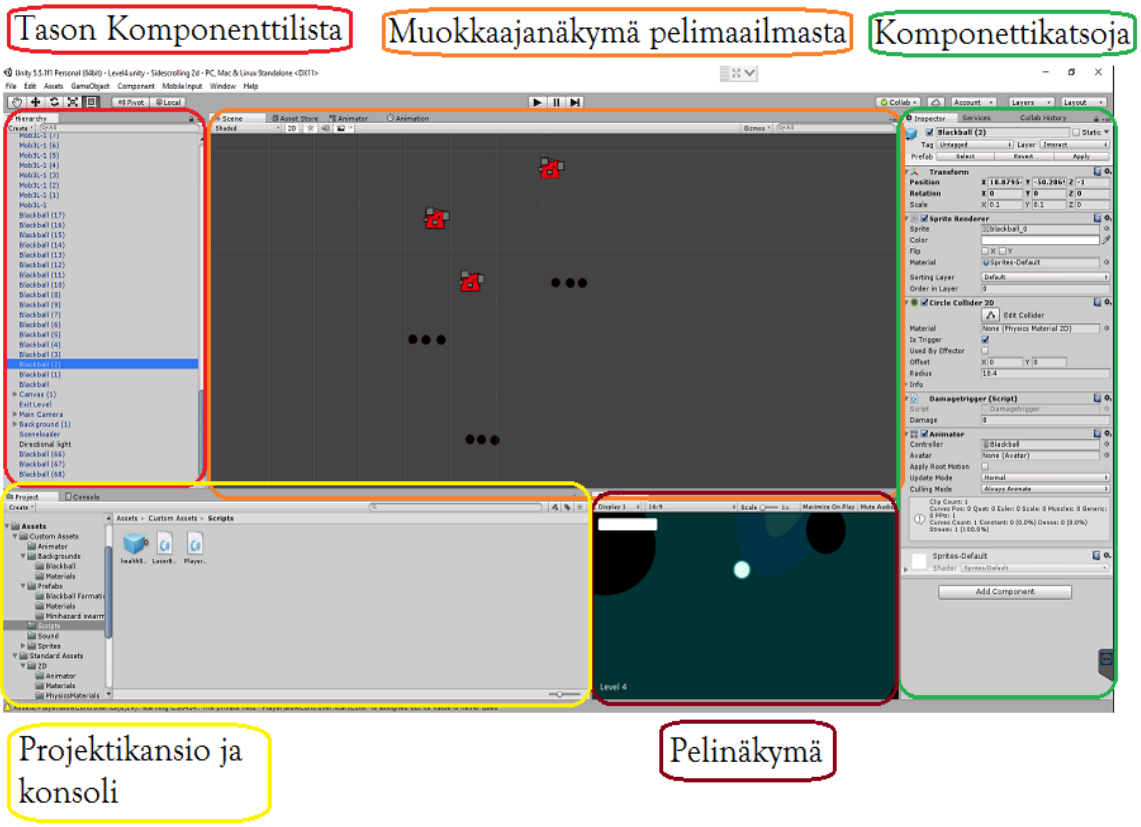
2 OHJELMOINTIMENETELMÄT

Unity ohjelmointiympäristö

Unity ohjelmointiympäristö on Unity Technologies yhtiön valmistama ja ylläpitämä ohjelmisto, joka on tarkoitettu peli- ja simulaatiokehitykseen. Sen ydin on kirjoitettu C++ ohjelmointikielellä, joka paljastaa .NET rajapinnan käyttäjille ja sen koodieditori on kirjoitettu C Sharp (C#) kielellä tämän päälle (Lucas Meijer 1. 2010). Vaikka alkujaan ohjelmaa oli sovellettu vain Applen Mac OS X käyttöjärjestelmille, se on myöhemmin laajennettu toimimaan useassa eri käyttöjärjestelmässä ja ohjelmaa voidaan käyttää valmistamaan pelejä kaikille suosituimmille pelilaitteille kuten Nintendon 3DS ja WII konsoleille ja Playstation 4 laitteistoille.

Ohjelmasta on tarjolla neljä eri tasoista versiota i) personal, ii) plus ja iii) pro ja iv) Enterprise versiot. Näistä ensimmäinen (i) on tarkoitettu harrastelijoille ja aloittelijoille, toinen (ii) edistyneemmälle pelinkehittäjälle ja kolmas on tarkoitettu ammattimaiseen (iii) pelienkehittämiseen. Yritys tarjoaa myös palveluna ns. ”enterprise” -versiota (iv), jonka he voivat muokata asiakkaan organisaation tai yrityksen tarpeiden mukaiseksi.

Unity tarjoaa nettisivuillaan myös ohjelman yhteydessä ns. ”asset store” palvelua, jossa sen käyttäjät voivat jakaa ja myydä valmistamiaan ohjelmakomponentteja (avoin koodi) tai ohjelmointiprojektien tuloksia esimerkiksi pelejä tai niihin soveltuvia osia.



KUVA 1. Unityn käyttämä koodieditori (kuva kaappaus).

Kuvassa 1 on Unityn koodieditori. Unityn editoria voi muokata haluamansa laiseksi, sen näkymän eri osia voi siirrellä ja niiden kokoja muuttaa vapaasti, keskellä on päänäkymä ja alhaalla oikealla on esinäkymä (pelinäkymä) siitä mitä pelin kamerakomponentti kuvaa alkuperäisessä sijainnissaan. Tämä Unityn valmis kamera (Camera) komponentti määrittää sen mitä pelaaja näkee pelistä tietokonenäytöllä.

Unity käyttää termiä ”asset” kuvaamaan kaikkia editorin käytössä olevia pelikomponentteja, editorissa peliin voi lisätä peliesineitä, joita se kutsuu nimellä ”Gameobject”. Tällainen uusi pelikomponentti sisältää valmiina vain sen paikan peliavaruudessa ja nimen. Näihin pelikomponentteihin voidaan sitten lisätä myös kaikki muut halutut ominaisuudet kuten kuvat, esim. pelihahmojen (2D/3D) graafinen ulkonäkö, animaatiot, äänet ja toimintoja ohjaavat koodinpätkät.

2.1. Ohjelmointikielet

Unity käyttää kolmea eri ohjelmointikieltä, C#, Unityscript ja boo, näistä C# ja boo ovat tavallisia kieliä ja Unityscript on Javascriptin muokattu versio. Unityn pääasiallinen hyvin aloittelijoille sopiva ohjelmointitapa on olemassa olevien käskysarjojen (skriptien) käyttö ja ohjelmointitaitojen kehittyessä uusien vastaavien tekeminen ja niiden kiinnittäminen pelin editorissa oleviin joko valmiisiin tai itse muodostettuihin peliesineisiin ja siten niiden toimintojen ohjaamiseen peliin soveltuvalla tavalla.

2.1.1 C Sharp (C#) ohjelmointikieli

C# on Microsoftin kehittämä kieli, joka sai nimensä musiikin terävä-C nuotin mukaan, koska terävä C on askeleen C:tä korkeampi (James Kovac 2007). Vaikka alun perin kieltä pidettiin java kopiona, sen version C# 2.0 myötä kielen kehitys on eronnut enemmän javasta ja C# 3.0 versioon lisätty tuki funktionaaliselle kielelle on sallinut keskittymisen enemmän ohjelmien logiikan kehitykseen (Hasan Noorul. 2012).

C# on unityn käytetyin kieli kattaen yli puolet sen käyttäjistä ja Unity3D 5.0 versiosta eteenpäin myös sen pääkieli, ennen tätä esimerkit ja dokumentaatiot tehtiin unityscript-kielellä ja käännettiin sen jälkeen C# ja boo kielelle. (aleksandr 2014).

2.1.2 Unityscript ohjelmointikieli

Unityscript muistuttaa javascriptiä, mutta niissä on useita eroja, jotka johtuvat siitä, että unityn kehittäjät muuttavat sen toiminnallisuutta vapaasti omien käyttötarkoitusperiensä mukaisesti. Tämä estää puhtaasti javascript pohjaisten ratkaisujen käytön unityn moottorissa.

Tärkeitä eroja javascriptiin on useampia mm. Unityn komentosarjojen luokallisuus, unityscript toimii luokkien perusteella ja pyrkii siten vähentämään kirjoitustarvetta olettamalla, että luokat jaetaan yksittäisiin tiedostoihin. Unity vaatii puolipisteet kaikkien koodirivien lopuksi mutta javascripti ei. (wiki.unity3d)

2.1.3 Boo ohjelmointikieli

Boo on pythonin kaltainen funktionaalinen ohjelmointikieli joka käyttää Microsoftin .NET tai linux MONO kirjastoja. Kielen käyttäjämäärä unitylla oli vähäinen, alle puoli prosenttia, mistä syystä 5.0 versiosta eteenpäin sen virallinen tuki lopetettiin ja nykyisin sitä ei voi käyttää ohjelmointikielenä unityn editorilla. (aleksandr 2014).

2.2. Työssä käytetyt muut välineet

Työhön liittyvät kuvat piirrettiin pääosin verkkoselaimessa toimivalla Piskel ja Windows Paint ohjelmalla. Pelin äänet äänitettiin Windows 10:n mikrofoniohjelmalla ja editoitiin Twisted Wave Online verkkoselain editorilla. Ohjelmointi tehtiin omalla Windows 10 pohjaisella tietokoneella tai käyttämällä sitä TeamViewer etäyhteys ohjelman kautta. TeamViewer on ohjelma, jolla voidaan ottaa etäyhteys toiseen tietokoneeseen. Ohjelman saa ilmaiseksi internetistä henkilökohtaiseen käyttöön (https://www.teamviewer.com/fi).

3 PELITUOTANNON PÄÄVAIHEET

Pelien kehitys ja tuotanto on monivaiheinen prosessi jota voi mallintaa eri tavoin, yksi yleiskattava tapa on jakaa pelin kehitysvaiheet neljään osaan. Nämä neljä osaa ovat konsepti, esituotantovaihe, tuotantovaihe ja jälkituotantovaihe. Joskus näitä tasoja laajennetaan tarpeiden mukaan esimerkiksi prototyyppivaiheella, joka tapahtuu esisuunnittelun jälkeen tai palveluvaiheella, joka taas tapahtuu pelin jälkituotantovaiheen jälkeen (Aalto O. 2015).

Pelin suunnittelu alkaa yleensä suunnitteluosuudella, joka koostuu konsepti- ja esituotantovaiheesta. Nämä molemmat vaiheet voidaan edelleen jakaa useampaan osaan esimerkiksi suuremmissa pelituotanto yrityksissä, poimien parhaita ideoita jatko työstettäväksi ja kehitettäväksi.

3.1. Konseptivaihe

Pelin konseptivaiheessa esitellään pelin idea ja tämä vaihe voi olla yksinkertainen tai monimutkainen, tässä vaiheessa mietitään, pohditaan, keskustellaan ja päätetään, millainen peli tuotetaan (Edwards R. 2006). Konseptivaiheen voi myös jakaa kolmeen osaan alustus, esisuunnitelma ja lopullinen suunnitelma, tätä tapaa suositaan suuremmissa peliyrityksissä ja niitä käytetään pelin idean esittelyyn (Laramee D.F. 1999).

Alustusvaiheessa valmistetaan lyhyt dokumentti, jossa esitellään pelin idea ja sen valmistuksen vaatimukset. Määritetään pelin pää-ominaisuudet ja tyylimaailma, kenelle peliä on tarkoitus myydä ja millainen kokemus pelin pelaajalle halutaan antaa, kootaan pelin valmistamiseen tarvittavat resurssit ja selostetaan miten kykenevä ja ammattitaitoinen olet tai tiimisi on tämän pelin valmistamiseen. (Laramee D.F. 1999).

Esisuunnitelma on laajennus alustusvaiheen määrittämään ideaan, tässä vaiheessa pelin tarkemmat tekniset vaatimukset, mekaniikat ja mahdollinen tarina ideoidaan ja hiotaan. Tämä sisältää esimerkiksi pelin erilaiset objektit, superliikkeet ja millaisia erilaisia ympäristöjä peli sisältää. (Laramee D.F. 1999).

Lopulliseen suunnitelmaan voidaan siirtyä, kun kaikki ovat tarpeeksi tyytyväisiä esisuunnitelmaan ja lopullisessa suunnitelmassa tähdätään mahdollisimman yksityiskohtaiseen kuvaukseen pelistä. Tässä dokumentissa kuvataan kaikki pelin osat, ominaisuudet, pelaajien kokemuksiin vaikuttavat tekijät ja pelin markkinointistrategia ja hinta. Ominaisuudet kuvataan tarkkojen algoritmien kuvauksin, määritellen pelitapahtumien tarkat ajoitukset ja mitä tietoa annetaan pelaajalle. (Laramee D.F. 1999).

3.2. Esituotantovaihe

Esituotantovaiheessa peli suunnitellaan, sen yleinen ulkonäkö, mitä pelissä voi tapahtua, miten se voitetaan ja miten sitä pelataan. Tässä vaiheessa selvitetään pelin mahdolliset rajoitukset, joko idean, laitteiston tai standardien takia. Pelin tasojen kartat ja taidetta valmistetaan kuvastamaan sen eri vaiheita ja nämä kaikki kootaan suunnitteludokumentaatioon. (Edwards R. 2006).

Tässä vaiheessa kootaan projektin tuotantosuunnitelma, jossa määritetään lopullisen suunnitelman ominaisuuksien implementointi, budjetti, grafiikat esimerkein ja laaditaan mahdollinen käsikirjoitus pelin tapahtumille. (Laramee D.F. 1999).

Tässä vaiheessa valmistetaan usein pala peliä tai sen prototyyppi, jonka on tarkoitus toimia pohjana pelin kehitykseen tuotantovaiheessa, toimien grafiikan ja tyylin mallina.

Esimerkkinä epäonnistumisista tässä vaiheessa on kuuluisa Mass Effect: Andromeda, jonka monista ongelmista yksi oli esituotantovaiheessa tekemättä jätetyt päätökset, kuten pelin kattavuus joka rajattiin vasta myöhään tuotantovaiheessa pelissä olevaan muutamaan planeettaan. (Schreier J. 2017).

3.3. Tuotantovaihe

Tuotantovaiheessa suurin osa peliin menevästä työstä tapahtuu, työn piiriin tuodaan lisää osaamista ja henkilöstöä tarpeen mukaan ja pelin suunnittelijat työskentelevät yhdessä graafikkojen, artistien ja ohjelmoijien kanssa, jotta pelin suunnitelma implementoidaan oikein. Haasteiden ilmestyessä, rajoitusten, toimimattomien mallien tai suunnitelmissa

olevien aukkojen takia, suunnittelijat joutuvat keksimään ratkaisuja ja uusia malleja. (Edwards R. 2006).

Pelin ympäristöt, eri mallit ja animaatiot valmistetaan graafikkojen ja artistien toimesta ja samalla ohjelmoijat valmistavat pelin moottorin joka huolehtii pelin graafisesta ulkonäöstä kuten valaistuksesta ja artistien/graafikkojen muotoilemien peliobjektien (tai muiden tuotosten) sijoittamisesta peliin. Ohjelmoijat luovat myös ”tekoälyn”, joka määrittää pelin eri komponenttien toimintaa ja sitä miten ne toimivat keskenään vuorovaikutuksessa. (Edwards R. 2006).

Kun pelin keskinäiset elementit on implementoitu, aletaan peliä muokata sen alustalle (mobiili peli, tietokonepeli, pelikonsoli peli) sopivaksi ja tehdään mahdolliset leikkaukset tai lisäykset pelin ulkonäköön, jotta se toimii kuhunkin pelialustaan sopivalla tavalla. (Edwards R. 2006).

3.4. Jälkituotantovaihe

Tämän jälkeen peli siirtyy jälkituotantovaiheeseen, jossa pelin alpha versio yleensä valmistetaan. Tämä vaihe koostuu pääosin pelin testauksesta ja kaikkien eri mahdollisten virheiden korjauksesta. Alpha versiossa pyritään tunnistamaan suurimmat virheet ja ongelmat korjausta varten. (Edwards R. 2006).

Korjausten jälkeen pelin beetaversio valmistetaan, tässä vaiheessa pelistä varmistetaan sellainen versio joka toteuttaa mahdolliset standardit ja tässä vaiheessa pyritään myös löytämään ja kirjaamaan ylös kaikki mahdolliset virheet vakavuusasteittain. Kun pelin tärkeimmät virheet ja mahdollisimman moni muu virhe on korjattu ja se täyttää kaikki standardit valmistetaan siitä lopullinen versio. (Edwards R. 2006).

Palveluvaihe koostuu pelin julkaisun jälkeisistä tapahtumista kuten päivityksistä, lisäsisällön tuottamisesta ja palvelimien ylläpidosta. Kyky päivittää pelejä julkaisun jälkeen on tullut yhdeksi tärkeäksi osaksi pelien suunnittelua, tämä sallii mahdollisten kriittisten virheiden korjauksen vielä pelin julkaisun jälkeenkin.

4 PELIN KONSEPTI JA TARKOITUS

4.1. Päämäärä

Pelissä on päämäärä päästä hyppimällä tasolta toiselle viimeisen pelikentän pohjalle. Peli muodostuu useasta valmiista pelikentästä, joissa hidasteiden ja vaarojen määrät lisääntyvät asteittain. Pelikenttien valaistus vähenee ja pelaajan valaisema alue pienentyy tasolta toiselle edettäessä, jolloin peli vaikeutuu asteittain. Mahdollisesti peliin lisättävä suoritusaikarajat pakottavat pelaajahahmon pysymään aktiivisena. Pelaajalla pystyy liikuttamaan pelihahmoja nopeasti sivuttain pyrkien näin auttamaan niitä väistämään eteen tulevia vaaroja ja pysymään jatkuvassa liikkeessä.

4.2. Pelin tausta tarina

Kauan sitten, maailman syvyyksistä alkoi ilmestyä Pimeyttä, valoa syövää pimeä voimakenttä, joka imi myös elämän ja sen rakennusaineet kaikesta mitä se koski. Ajan kuluessa pimeys levisi laajemmalle ja laajemmalle, aiheuttaen ihmiskunnan pakenemisen aina entistä korkeammille paikoille turvaa etsimään. Ihmiskunnan viimeiset hetket ovat nyt lähestymässä, korkeammalle ei enää voi paeta ja viimeinen toivo sijaitsee nyt taikureiden viimeisessä yrityksessä saattaa valopallo, jonka tehtävä on hakeutua pimeyden synkimpään ytimeen ja tuhota se.

4.2.1 Aloitus

Pelin alkaessa pelaajalle kerrotaan, että maailma on tuhon partaalla ja ainoa tapa pelastaa se on viedä valopallo pimeyden ytimeen. Peli alkaa aloitusnäkymän jälkeen ensimmäiseltä tasolta, jossa ei vielä ole vaaroja, pelaaja esitellään erilaisille vaaroille muutama kerrallaan ja niiden määrää aletaan asteittain pelin edetessä lisäämään.

4.2.2 Päätös

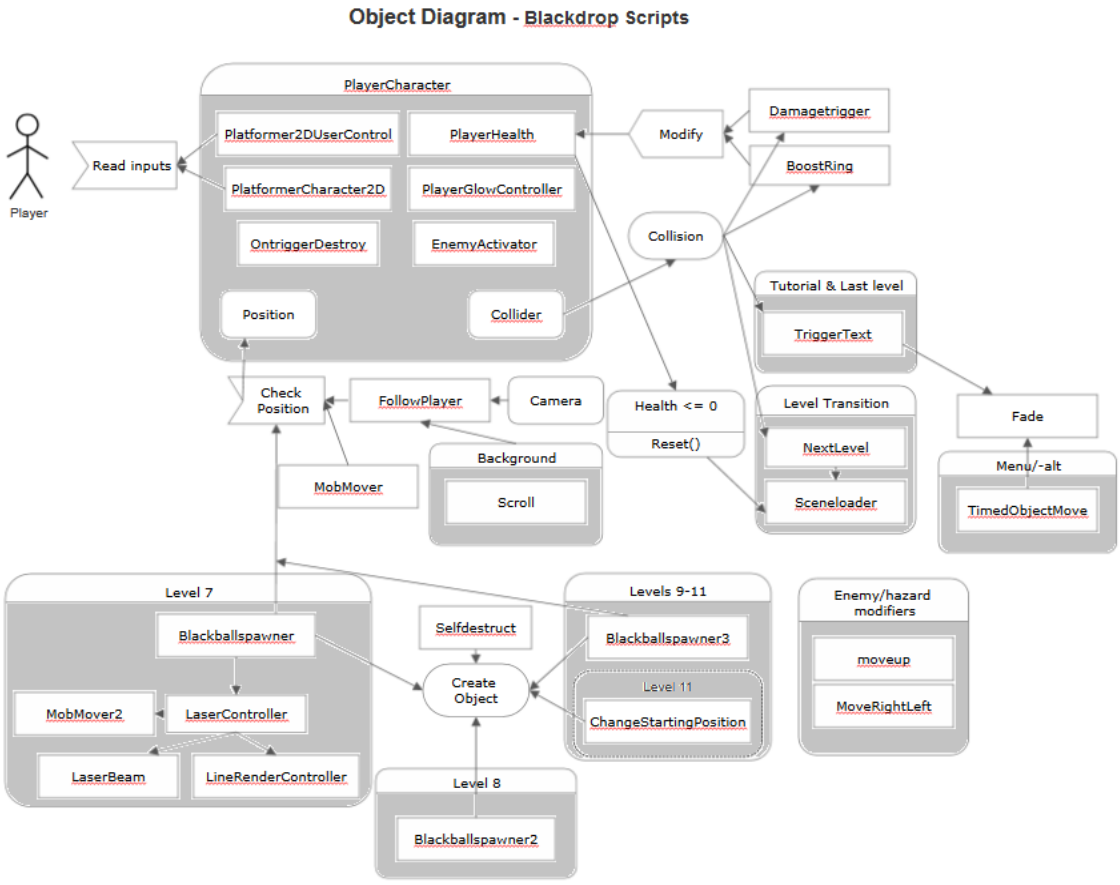
Pelaaja saapuu viimeisen kentän pohjalle, siellä hän näkee ilmassa leijuvan kivi korokkeen, jonka alla on synkintä pimeyttä, jota hänen valonsa ei enää valaise.

Korokkeella hän näkee laatikon, josta edellä mainittua pimeyttä tihkuu ja jonka yläpuolella ovat metalliset pidikkeet, joihin hänen valonsa sopii. Pelaaja hyppää koroketta kohti, yrittäen laittaa valon pidikkeisiin ja vain onnistunut yritys, aiheuttaen valoräjähdyksen joka valkaisee näyttöruudun. Tämän jälkeen peli tallentaa tämän tapahtuman muistiin ja palaa aloitusruutuun, joka on nyt valaistu.

5 PELIN RAKENNE JA OHJELMOINNIN ERI INKREMENTIT

5.1. Pelin rakenteen kokonaisprosessi kaavio

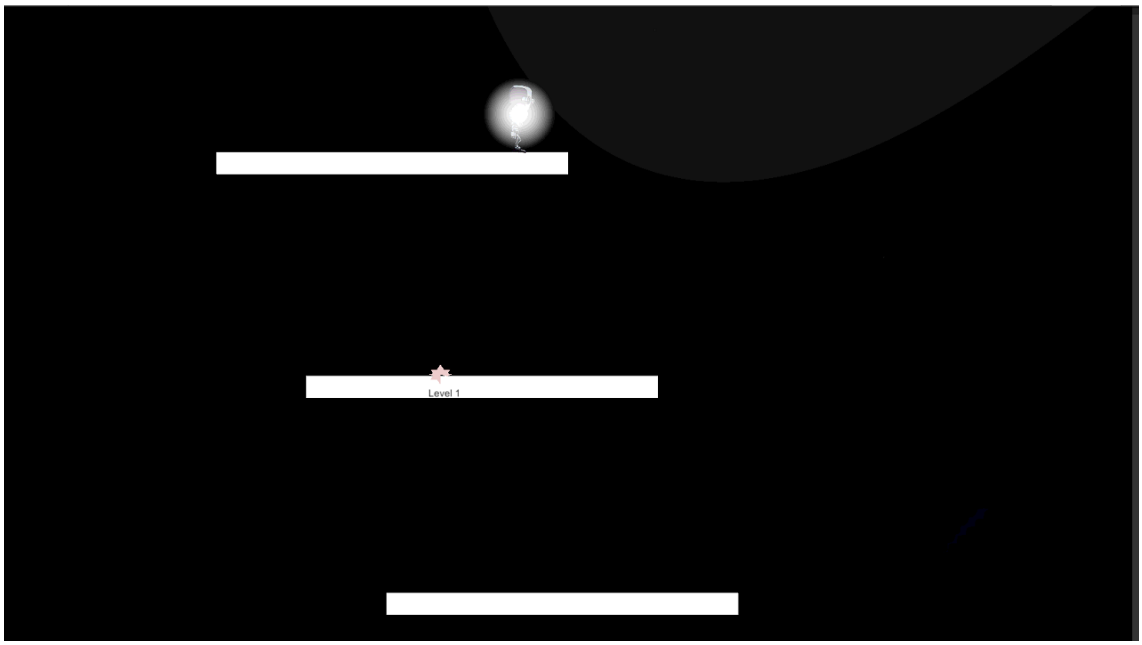
Kuvassa 2 on yhteenveto pelin toiminnoista, nimetyt suorakaiteet kuvataan alla olevassa tekstissä yleisesti ja harmaapohjaisissa käskysarjakuvissa yksityiskohtaisesti, kuvan muolet yhdistävät toisiinsa liittyviä käskysarjoja ja määrittävät tapahtumien kulkusuuntaa (ohjaavat pelin kulkua). Kuvassa nimetyt isommat harmaapohjaiset laatikot määrittävät niiden sisältämien käskysarjojen toimialueita. Esimerkiksi kuvan PlayerCharacter laatikon sisällä ovat kaikki ne käskysarjat, jotka ovat osa pelaajahahmon määritystä.



KUVA 2. Kaaviokuva pelin käskysarjoista ja miten ne toiminnallisesti liittyvät toisiinsa.

5.2. Inkrementti 1

Pelin ensimmäinen inkrementti, ja sen konseptivaihe, muodostui ohjelmaan tutustuessani ja valmistin sen pääosin unityyn sisältyvillä valmiilla pelikomponenteilla. Ts. käyttämällä sen valmista kaksiulotteista pelaajahahmoa ja lisäksi tein testikenttään erilaisia laattoja (KUVA 3). Sijoitin laatat testikenttään siten, että niiden avulla hahmo pystyisi jatkamaan putoamista pelin alusta sen loppuun asti ja, että se vaatisi aktiivista pelaajahahmon liikuttamista ja laattaesteiden väistämistä.



KUVA 3. Testikenttä Unity editorissa.

Lisäsin myös testikenttään liikkuvan taustakuvan paikan (quad). Unityssä Quad komponentti termi tarkoittaa litistynyttä 3D suorakaidemaista kappaletta, jonka x-y-z akseliston dimensioita voidaan säädellä ja z akseli pituuden lähestyessä 0 se litistää suorakaiteen kaksiulotteiseksi. Tätä komponenttia voidaan käyttää useisiin eri tarkoituksiin, mutta tässä käytin sitä kuvan kiinnityspaikkana testikentässä. Tätä kuvaa voidaan helposti myös vaihtaa tarvittaessa pelin vaatimalla tavalla ja laitoin tämän kuvapaikan kiertämään tätä pintaa, joka luo visuaalisen illuusion itseään toistavasta taustakuvasta. Tämä liike saatiin aikaan alla olevassa kuvassa olevalla C# kielisellä käskysarjalla 1 (Kuva 4).

```
public class Scroll : MonoBehaviour {
    //Change scrolling speed
    public float speed = 0.2f;

    // Update is called once per frame
    void Update () {
        //Create Vector2 object with y coordinate
        //change by the time since the start of the game times the speed value.
        Vector2 offset = new Vector2(0, Time.time * speed);
        //Get the rendere of this object and change Offset value
        GetComponent<Renderer>().material.mainTextureOffset = offset;
        }
}
```

KUVA 4. Käskysarja 1. Kuvan pyörityskoodi (Charger Games. 2015).

5.2.1 Hahmoa seuraava kamera

Tässä kohtaa käytit apuna Unityn Kameraa (Camera) komponenttia. Tämä komponentti määrittää sen mitä pelaaja näkee pelistä tietokonenäytöllä. Tässä ohjasin kameran liikettä käskysarjalla nro 2 (Kuva 5). Halusin, että kamera seuraa pelaajaa vain alas ja ylöspäin suuntautuvissa liikkeissä (ei sivuille). Pelaajahahmo ei saa myöskään poistua kameran näkyvyysalueelta, tämä estettiin laittamalla palkkimaiset rajoittimet kameran näyttämän alueen oikeaan ja vasempaan laitaan, nämä rajoittimet asetettiin kameran alaisiksi, (child) jolloin ne liikkuvat kameran mukana.

```
void Update () { // Update is called once per frame
    //Change position every frame
    //to match the followed objects position
    //modified by offset and DistanceAway value
    moved.transform.position =
    new Vector3(moved.position.x, followed.position.y+offset,
    followed.position.z - distanceAway);
    }
```

KUVA 5. Käskysarja 2. FollowPlayer.cs Update() funktio.

Käskysarja 2 käyttää aloituksessaan tehtyä viittausta itseensä ja muuttaa editorissa valitun pelikomponentin y- ja z-akselin sijaintia vastaavasti, kuin seuratun pelikomponentin sijainti, näitä arvoja muokataan offset ja distanceAway arvoilla, x-akseli pidetään samana. Update funktio on yksi unityn valmiista toiminnallisuuksista, se toistaa itseään joka kerta kun peli piirretään näytölle ja se on hyvä tapa toistaa käskysarjoja tahdissa pelin kanssa.

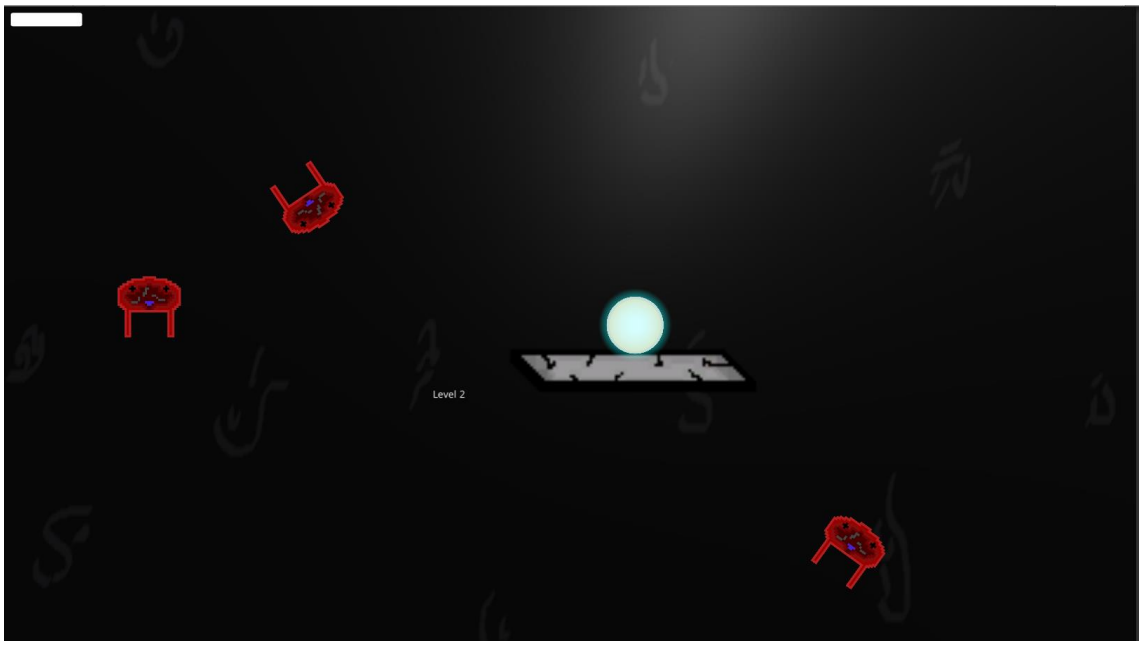
5.2.2 Esiin tulleet jatkokehitystä vaativat haasteet

Ensimmäisen pelin inkrementin ongelmat liittyivät pelihahmon uppoamiseen laattoihin ja kameran ongelmiin. Kamera kuvaa ja näyttää pelimaailman objekteja eri perspektiiveistä, liikkuvan taustakuvan kanssa ilmestyi ongelma siinä, että asiat jotka peli piirsi aikaisemmin, kuten laatat, ylikirjoitetaan jolloin ne katoavat pelaajan näkyvistä. Pelihahmo myöskin pudotessaan tarpeeksi nopeasti menee laattojen sisälle ja tämä joissakin tapauksissa estää hyppäämisen.

Mahdollisia ratkaisuja laattojen katoamiseen voisi olla niiden uudelleenpiirtäminen pelaajanhahmon lähettyvillä uudella käskysarjalla, joka liikuttaa niitä. Laattojen läpi putoamiseen ei ole vielä ratkaisua, mutta kokeilen jos laattojen uudelleen teolla voidaan ratkaista havaittu ongelma.

5.3. Inkrementti 2

Toisen inkrementin alussa halusin lisätä peliin koodin, joka saa aikaan siirtymisen pelin tasojen eli kenttien välillä, lisätä pelaajanhahmon elämäpisteet jotka kuvaavat pelihahmon vahingonsietokykyä ja parantaa pelissä vastaantulevien esteiden ulkonäköä. Lisäsin peliin sen aloitusikkunan (ulkonäön), koodasin esteiden ja vihollisten parannetut ulkonäöt ja vihollisten toimintaa ohjaavan käskysarjan. Käytin näitä koodaamiani uusia elementtejä ja niiden avulla loin pelin toisen inkrementin (pelitason). Alla kuva sen kentästä numero kaksi (ks. Kuva 6).



KUVA 6. Pelikehityksen toisen inkrementin toinen taso. Kuvassa punaiset meduusat ja pelaajahahmo (valkoinen pallo).

Kuvassa näkyvä valopallo on pelaajahahmon uusi ulkonäkö, ruudun vasemmassa yläreunassa olevan valkoisen palkin leveys toimii nyt elämäpisteiden (player health) määrän osoittimena. Elämäpiste mittarin muodostavan käskysarjan pohjan hain unityn valmiista käskysarjoista ja muokkasin sitä tähän peliin soveltuvaksi. Sen jälkeen kirjoitin käskysarjan (Kuva 7), joka vähentää pelaajahahmon elämän pisteitä sen kohdatessa esteitä tai vihollisia pelin aikana.

```
public class Damagetrigger : MonoBehaviour {
        //Field for controlling damage taken
        [SerializeField]
        private int damage = 3;

        //When a collider on this object touches another collider or rigibody
        private void OnTriggerEnter2D(Collider2D other){
        //If the other object has the tag Player attached
        if (other.tag == "Player")
        //Get the PlayerHealth component of that
        //and use it's TakeDamage function
        other.GetComponent<PlayerHealth>().TakeDamage(damage);
        }
}
```

KUVA 7. Käskysarja 3. Kutsuu TakeDamage funktiota PlayerHealth käskysarjasta

Kuvan 7 käskysarja käyttää Collider2D komponenttien reaktiotoimintoa toisen vastaavan pelikomponentin kohtaamiseen pelikentällä, jos tämän koodin käyttämä komponentti on asetettu liipaisimeksi (trigger), tarkoittaen sitä, että se ei estä muiden Colliderin sisältävien komponenttien kulkua lävitsensä. Colliderin sisältävät pelikomponentit reagoivat Colliderille määritetyn toiminta-alueen sisällä toisien saman tyyppisten komponenttien kanssa usealla eri tavalla ja niillä on useita vaihtoehtoisia valmiita reaktiokäskysarjoja. Niiden pääasiallinen tarkoitus on säädellä peliesineiden kohtaamisen luonnetta eri tavoin.

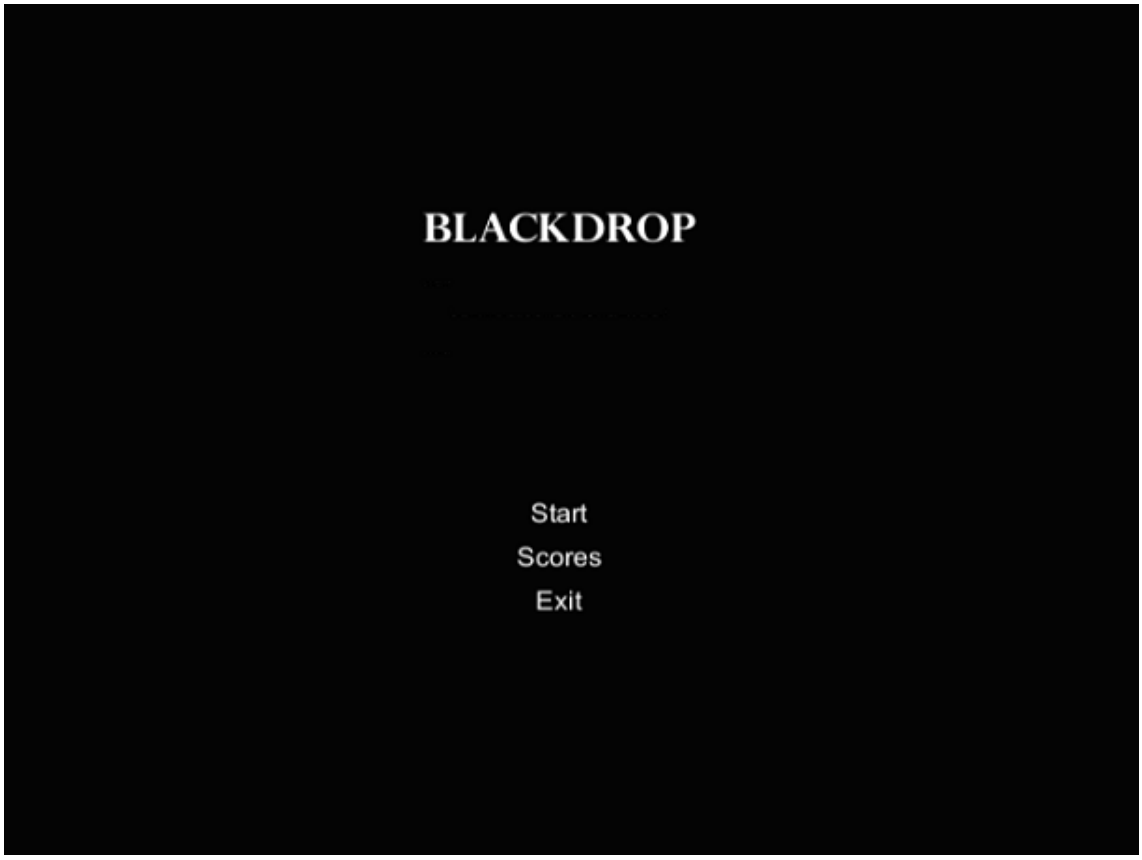
Yksi liipaisintyyppinen Collider2D lisättiin seuraamaan pelaajahahmoa, sen tehtävä on lähettää tuhoa-komento (Destroy()) sen toiminta-alueelle tuleville pelikomponenteille. Näin se poistaa ohimennyttä materiaalia pelikentältä ja pysäyttää näin myös aikaisemmin aktivoituneiden vihollisten käskysarjojen toiston.

Inkrementin kaksi loppuun löysin myös sopivan käskysarjan joka toteuttaa tasosiirtymän eli pelikentän vaihtumisen toiseen (Pettit, N. 2005), tässä inkrementissä tasosiirtymää ei

vielä saatu tapahtumaan automaattisesti pelikentän lopussa, mutta sain sen toimimaan erikseen manuaalisesti nappia painamalla.
Edellisen inkrementin ongelman taustankuvan näkyvyydessä korjasin käyttämällä kahta kamera pelikomponenttia yhtä aikaa, siten että ensimmäinen näkee vain taustan ja toinen kaiken muun.

5.3.1 Aloitusikkuna

Pelin aloitusikkunassa ”BLACKDROP” teksti on näkymän keskellä. Aloitusikkunaan (Kuva 8) ohjelmoitiin tapahtuma, jossa pelin nimi siirtyy näytön keskeltä neljänneksen päähän näytön yläreunasta ja nimitekstin liikkuessa pelin valikon keskelle ilmestyy toiminnallinen ”start/exit” valikko.



KUVA 8. Pelin aloitusikkunassa ”BLACKDROP” teksti on näkymän keskellä ja aloituksen jälkeen se siirtyy kuvassa näkyvään muotoon.

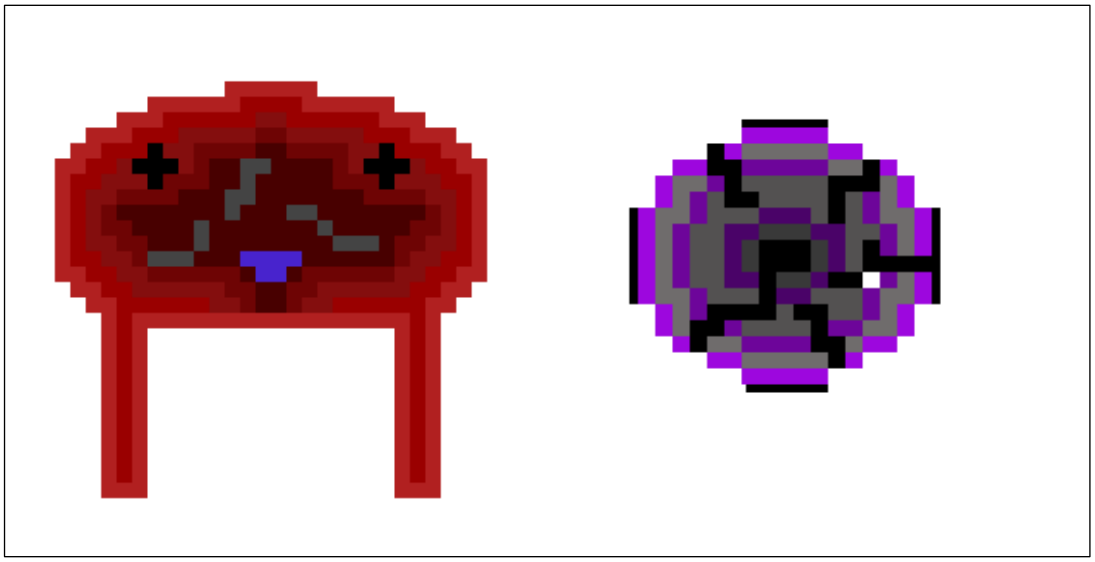
Nimiteksti aloittaa keskeltä ruutua ja siirtyy käskysarjan avulla neljänneksen päähän ruudun yläreunasta, samalla valikon tekstit ilmestyvät näkyviin, käyttäen kuvan 9 käskysarjaa, joka muuttaa tekstin värin näkyvyys arvoa (alpha) nollasta maksimiin eli täysin näkyväksi. Käskysarjaa käytetään antamalla sille aika t, joka on haluttu aika muutoksen kestolle ja viittaus muutettavaan pelikomponenttiin i, komennolla on vastinpari joka puolestaan haalistaa tekstin näkymättömäksi.

```
public IEnumerator FadeTextToFullAlpha(float t, Text i)
{
        i.color = new Color(i.color.r, i.color.g, i.color.b, 0);
        while (i.color.a < 1.0f)
        {
        i.color = new Color(i.color.r, i.color.g, i.color.b, i.color.a +
        (Time.deltaTime / t));
        yield return null;
        }
}
```

KUVA 9. Käskysarja 4. Tekstin näkyvyyden muutos näkyväksi (LeftyRighty. 2012)

5.3.2 Uudet vihollisyksiköt

Pelin tähän inkrementtiin valmistin kaksi toiminnaltaan erilaista vihollisyksikköä (Kuva 10), punaisen meduusan ja purppuran pallon. Meduusan on tarkoitus liikkua tiettyä liikerataa pitkin ja pallon seurata pelaajaa. Kuvat valmistettiin PISKEL –ohjelmalla (ks. piskelapp.com).



KUVA 10. Vihollisyksiköt - punainen meduusa ja purppura pallo

MobMover.cs käskysarja sisältää kolme numeroitua vaihtoehtoista ohjetta siitä, miten pelikomponenttia pelikentällä liikutetaan. Ensimmäinen ohje liikuttaa pelikomponenttia (punainen meduusa) kolmiomaisesti kolmen määritetyn pelikentän pisteen välillä. Toinen liikuttaa pelikomponenttia (purppura pallo) jatkuvasti pelaajahahmoa kohti. Kolmas ohje (Kuva 11) puolestaan tarkistaa määritetyin aikavälein pelaajan sijainnin eli seuraa pelaajan liikettä ja siirtää pelikomponenttia viimeksi havaittua pelaajan sijaintia kohti (siis seuraa pelaajaa viiveellä). Pelikomponenteille on asetettu edellä mainituista vaihtoehtoisista ohjeista yksi unityn editorissa (public int muuttuja) ja valittu ohje määrittää kunkin pelikomponentin toimintaa.

```
if (m_timer <= 0)
        {
        targetPosition1 =
        GameObject.FindWithTag("Player").transform.position;
        transform.position = Vector3.Lerp(
        transform.position, targetPosition1, Time.deltaTime * speed);
        m_timer = 2f;
        }
 else if(m_timer >= 0)
        {
        transform.position = Vector3.Lerp(
        transform.position, targetPosition1, Time.deltaTime * speed);
        m_ timer -= Time.deltaTime;
        }
```

KUVA 11. Käskysarja 5. MobMover.cs käskysarjan kolmas toimintaohje (esimerkki)

Pelikomponenttien siirtymiset saatiin aikaan unityn komentokirjastosta löytyvää lerp käskyä käyttäen, joka määrittelee pelikomponentin liikenopeuden.

5.4. Inkrementti 3

Pelin kolmannessa inkrementissä valmistin sen lopulliset pelikentät yhdestä kahdeksaan asti ja viimeisen pelikentän, voittoruudun sekä ohjetason (tutoriaali), joka opettaa ja kertoo sinulle pelin ohjausnäppäimien käytön. Toiseen kamera pelikomponenteista lisäsin myös äänilähde pelikomponentin tämän soittaa pelinääniraitaa eli taustamusiikkia sen edetessä. Tätä pelin taustamusiikin toteutukseen käytettyä ääniraitaa voi vaihtaa editorissa, jos pelin ohjelmoija sitä haluaa.

Pelaajahahmoon kiinnitettiin käskysarja, joka muuttaa sen valokomponentin laajuutta ja kirkkautta ajan funktiona. Lisäsin tässä vaiheessa siihen myös valojälkikomponentin, joka piirsi värillistä linjaa sen perässä. Korvasin tämän myöhemmin paremman näköisillä pienillä valopalloilla, jotka löysin Unitin valmiista kuvista.

Tässä vaiheessa peliä tein myös uuden vihollistyypin (siniset rauskut), jotka käyttävät aikaisemmin mainittua käskysarjaa 5, joka tarkistaa pelaajan sijainnin tietyin aikavälein. Tämän lisäksi aloitin uuden lasersäteitä käyttävän vihollisen suunnittelun, joita pelaajan täytyy väistellä pelin edetessä.

5.4.1 Liikuttajat

Pelikomponentteja liikuttamaan luotiin kaksi komentosarjaa, ensimmäinen liikuttaa niitä edestakaisin vasemmalta oikealle ja toinen ylös tai alas. Molemmissa komentosarjoissa on editorissa muutettava nopeusarvo, jolla voi säätää pelikomponenttien liikenopeutta. Näitä kahta liikuttaja komentosarjaa moveup.cs (Kuva 12) ja MoveRightLeft.cs käytettiin Fallingblackball ja Sidewaysfallblackball komponenttien osana, niihin lisättiin myös selfdestruct.cs koodi joka tuhoaa komponentin etukäteen asetetun ajan kuluttua.

```
public class moveup : MonoBehaviour {
        public float speed = 5f;
        [SerializeField]
        private bool active = false;
        private float oldpos;
        // Use this for initialization
        void Start () {
        oldpos = GameObject.FindWithTag("Player").transform.position.y;
        }
    // Update is called once per frame
    void Update () {
        if (active) {
        this.transform.position = new Vector2(this.transform.position.x,
        this.transform.position.y + Time.deltaTime * speed);
    }
        public void Activate(){ //Public void to activate
        print("moveup activated");
        active = true;
        }
}
```

KUVA 12. Käskysarja 6. moveup.cs

Käskysarja sisältää loogisen boolean operaattorin (active), jonka ollessa epätosi (false) se estää tapahtuman ja, jos sen asetetaan todeksi (true), se liikuttaa pelikomponenttia y-akselilla sille annetun nopeusarvon mukaisesti. Lisäksi koodissa on Activate() funktio, joka muuttaa em. loogisen operaattorin epätodesta todeksi.

Käskysarjan 7 (Kuva 13) active operaattorin ollessa tosi, se siirtää pelikomponenttia x-akselilla toisen loogisen boolean operaattorin ’changedirection’ määrittämään suuntaan. Mikäli pelikomponentin x-akselin arvo ylittää 17 tai alittaa -17, ’changedirection’ arvo vaihdetaan ja komponentin liikesuunta muuttuu.

```
if (active) { //if active
if (!changedirection){ //change position of this object to the right
        this.transform.position = new Vector2(this.transform.position.x +
        (Time.deltaTime + 0.1f) * speed, this.transform.position.y); }

else if (changedirection) { //change position of this object to the left
        this.transform.position = new Vector2(this.transform.position.x -
        (Time.deltaTime + 0.1f)* speed, this.transform.position.y); }

//Change movement direction if the x of this object is too big or small
if (this.transform.position.x > 17 & !changedirection) {
changedirection = true; }
else if(this.transform.position.x < -17 & changedirection) {
changedirection = false; }  }
```

KUVA 13. Käskysarja 7. MoveRightLeft.cs if (active) osio

5.4.2 Ohjaavat komentosarjat

Taso seitsemän sisältää useamman liikkuvan osan ja näitä tapahtumia koordinoimaan luotiin kaksi komentosarjaa Lasecontroller.cs (Kuva 14) ja Blackballcontroller.cs. LaserBeam.cs funktio määrittää laseri sijainnin ja luo ruudulle punaisen laserimaisen ilmiön.

```
void Update () {  // Update is called once per frame
    if (GetComponent<Blackballspawner>().part == 3) { part = 3; }
    if(part == 1) { //If part 1
            if (timer > 0){ lineon = true; //if timer above 0
                timer = timer- Time.deltaTime; }
            else if (timer < 0){ lineon = false; // If timer belov 0
                laseron = true; part = 2;  timer = 10f; } }
    else if(part == 2){ // if part 2 //stop lasers, change mob positions
        if (timer < 0 & timer > -3){ laseron = false;
        GameObject.Find("BiggestMob-3").transform.localPosition =
        new Vector2(12f, 0);
        GameObject.Find("BiggestMob-3 (1)").transform.localPosition =
        new Vector2(-18f, 0);
        GameObject.Find("BiggestMob-3 (2)").transform.localPosition =
        new Vector2(2f, 0);
        GameObject.Find("BiggestMob-3 (3)").transform.localPosition =
        new Vector2(-8f, 0);   }
        else if (timer < -5 & timer > -8) { lineon = true; }
        timer = timer - Time.deltaTime; }//if timer condition, draw lines
        else if(part == 3){ //if part 3
        if (timer < -10){ //timer small, stop lines, activate laser
                lineon = false; laseron = true; }
        if (timer < -15){ //if timer smaller, stop laser, start lines
                laseron = false; lineon = true; timer = -5; }
        timer = timer - Time.deltaTime; } }
```

KUVA 14. Käskysarja 7. LaserController.

LaserController.cs komentosarjan tehtävä on määrittää milloin laserit voivat vahingoittaa pelaajahahmoa, käskysarjassa on kaksi osaa, laseron ja lineon, joita LaserBeam.cs käskysarjat seuraavat ja käyttävät laserien toiminnan koordinoimiseen. LaserController.cs komentosarjan myös muuttaa neljän BiggestMob-3 pelikomponentin sijaintia.

Blackballspawner käskysarja luo peliin putoavia mustia palloja pelaajahahmon yläpuolelle, tähän se käyttää erilaisia funktioita (ks. alla). Spawner funktio (Kuva 15) vastaanottaa muilta pelin funktioilta x-akselin sijainnin ja sitten katsoo pelaajan sijainnin y-akselilla ja luo mustan pallon tämän yläpuolelle.

```
void Spawner(int m_point){
//Creates blackball objects in passed position x
var gameObject = goref; //reference to gameobject in editor
pos = playerpos.position; //save playerposition to variable
pos = new Vector3(m_point, pos.y + 9, pos.z);// and change it
gameObject.transform.position = pos; //give pos to gameObject
Instantiate(gameObject); //spawn gameObject
}
```

KUVA 15. Käskysarja 8. Spawner funktio.

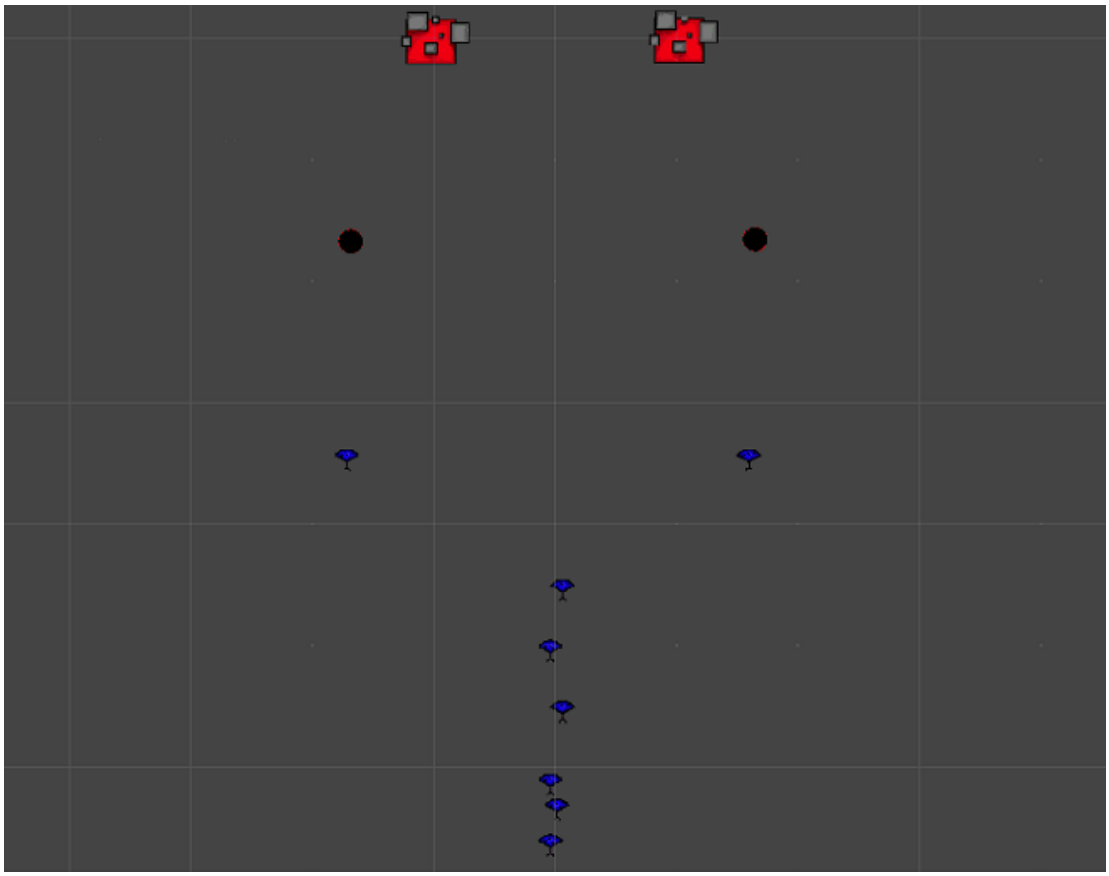
Spawner funktiota käyttää, muun muassa, neljä erilaista Cycle funktiota (Kuva 16), nämä funktiot toimivat yhdessä timer nimisen ajastin muuttujan kanssa. Cycle funktioista ensimmäinen antaa Spawner funktiolle sijainnit x-akselilla oikealta vasemmalle pelikentän halki, jättäen tyhjän kulkukohdan pelaajahahmolle pelinäkymän keskivasemmalle alueelle. Toinen Cycle funktioista toimii vastaavasti, mutta lähettää x-akselin sijainteja molemmilta reunoilta keskelle yhtä aikaa ja jättää pelaajahahmolle tilaa pelikentän keskelle. Kolmas Cycle funktioista aloittaa x-akselin sijaintikohdan lähettämisen keskeltä ja lähetetyt kohdat siirtyvät siitä vasemmalle ja oikealla jättäen tyhjää tilaa pelikentän reunoille ja neljäs Cycle funktio lähettää x-akselin sijainteja oikeasta reunasta vasempaan mutta jättäen välit joista pelaaja voi ketterästi pelaajahahmoa ohjaamalla päästä lävitse ilman että pelaajahahmo menettää elämäpisteitä.

```
void Cycle1(){ //Right to left, leave hole on left side
        Spawner(point);
        point -= 2;
        if (point == -4){ timer = 1f; point -= 4;}
        if (point < -19){timer = 5f; point = 19; cycle = 2; } }
void Cycle2(){ //Side to middle, hole in middle
        Spawner(point);
        Spawner(-point);
        point -= 2;
        if (point == 11) {timer = 1f; point -= 2; }
        else if (point == 1){timer = 5f; point = 0; cycle = 3; } }
void Cycle3(){ //Middle to sides, hole close to edges
        Spawner(point);
        Spawner(-point);
        point += 2;
        if (point >= 18){timer = 5f; point = 18; cycle = 1; part++;} }
void Cycle4(){ //Left to right with wide gaps
        Spawner(point);
        point -= 6;
        if (point < -20){point = 20; timer = 2.5f; } }
```

KUVA 16. Käskysarja 9. Cycle funktiot

5.4.3 Lopputulos

Pelin alussa menu ikkunan/aloituksen jälkeen tason yksi haasteet ovat mustat pallot ja punaiset villikarjut. Alussa yksinkertainen pelikenttä tutustuttaa pelaajan pelin toimintaan. Tasossa kaksi loppupuolella on suuri määrä mustia palloja joiden ohi voi kulkea vain kapeaa kaistaa pitkin. Taso kolme, kuva 17 alla, sisältää siniset viholliset (rausku), jotka aktivoituessaan paikallistavat pelaajan sijainnin ja lähestyvät pelaajaa nopeasti. Niiden väistäminen voi olla hankalaa mutta niiden tekemä vahinko eli elämäpisteiden menetys on vähäistä.

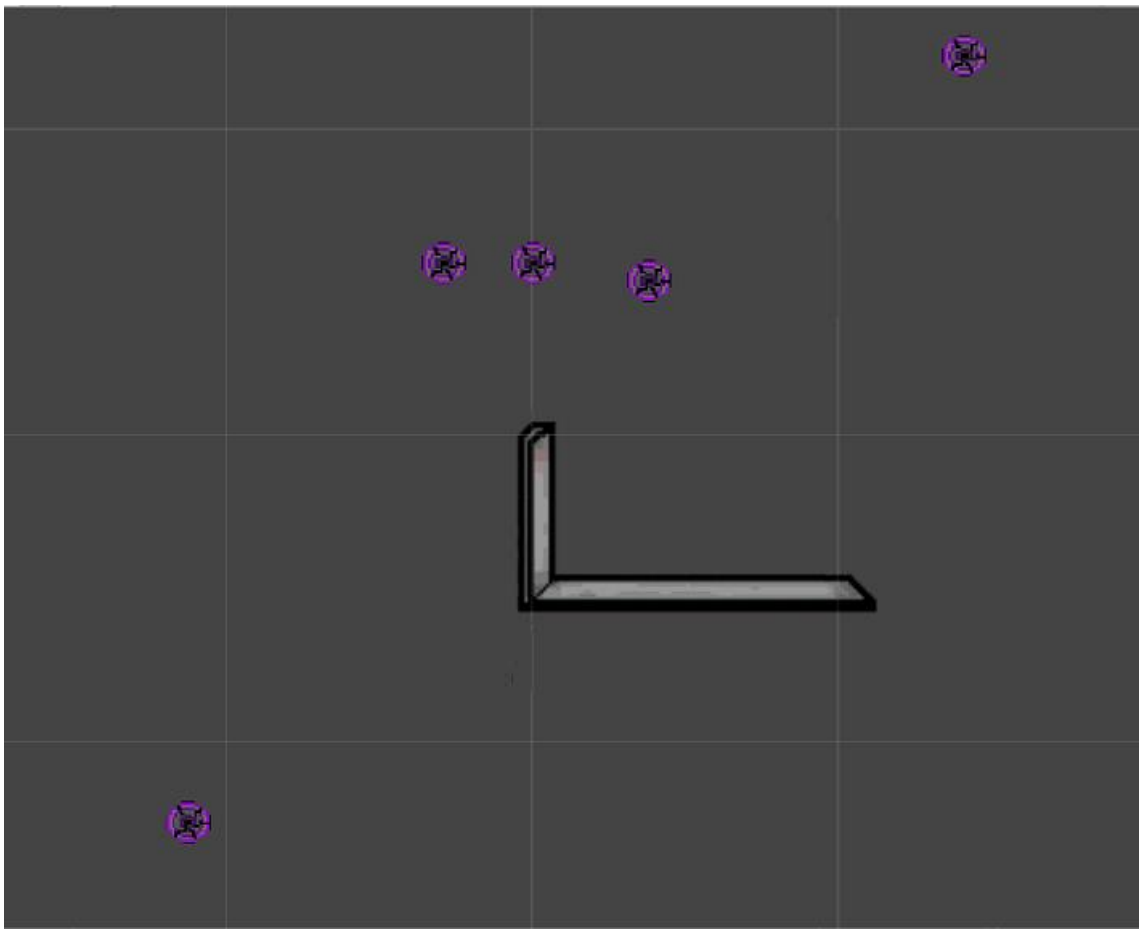


KUVA 17. Kuvakaappaus tasosta 3 editorissa, kuvassa sinisiä (rauskuja) ja punaisia (villikarjuja), sekä punareunaisia pallosalamoita joihin törmätessä pelaajahahmo menettää osan elämäpisteistään.

Neljäs kenttä on yhdistelmä näitä kolmea estetyyppiä (rausku, villikarju ja pallosalama). Näistä villikarju pelikomponentteihin olen kiinnittänyt MoveRightLeft.cs komentosarjan, joka liikuttaa niitä x-akselilla edestakaisin vasemmalta oikealla.

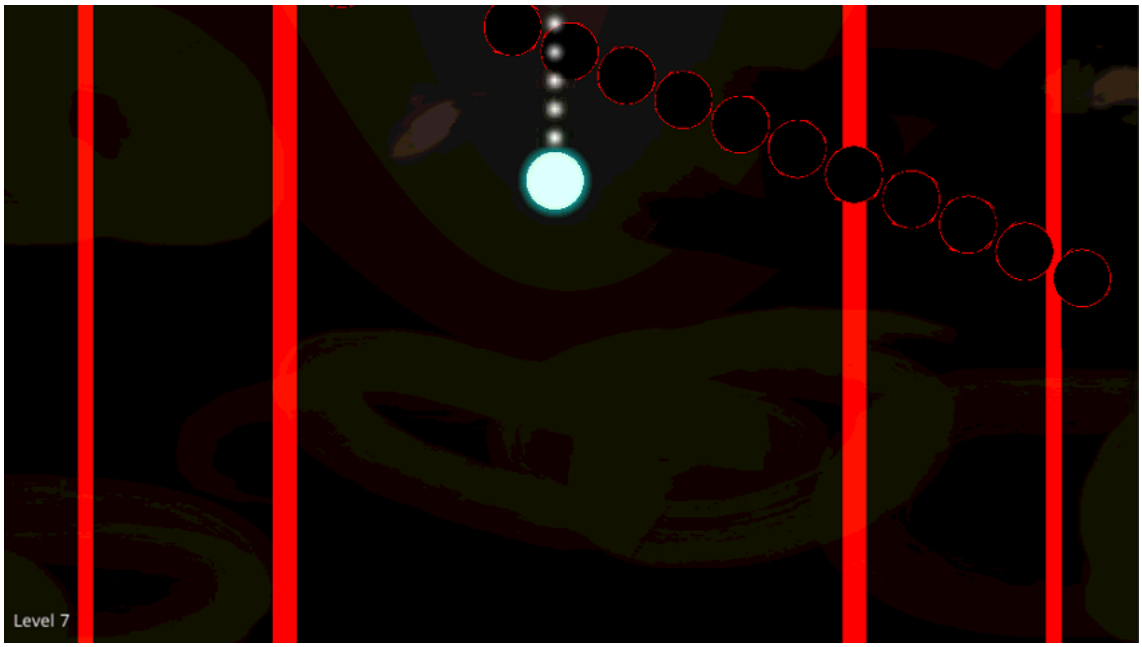
Tasossa viisi (Kuva 18) esitellään ensimmäisen kerran purppurat vihollispallot jotka pyrkivät seuraamaan pelaajaa tasaisella nopeudella ja vaihe jossa pelikentässä on useampi liikkuva villikarju, jotka toimivat samoin, kuin edellisellä tasolla.

Taso kuusi eroaa tasosta 5 siten että siinä käytettiin ensimmäisen kerran kaikkia esiteltyjä valmiita pelihahmoaja ja muita pelikomponentteja.



KUVA 18. Kuvakaappaus tasosta 5. Kuvassa purppuroita vihollispalloja joihin törmääminen aiheuttaa pelaajahahmolle sen elämäpisteiden menetyksen. L-muoto on satunnainen este.

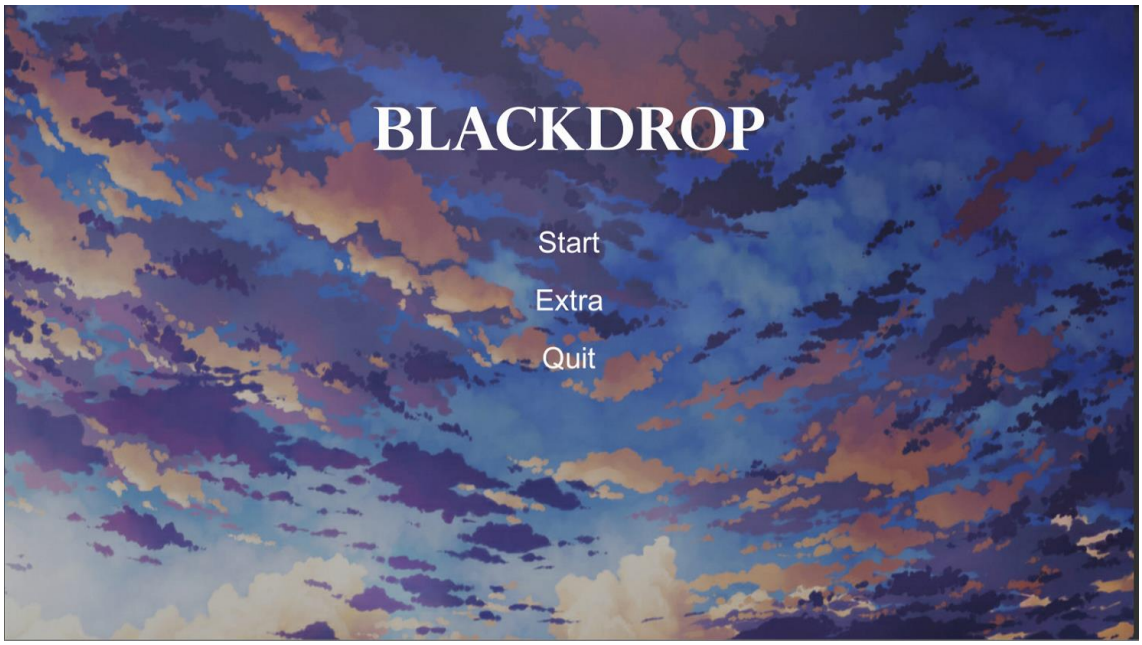
Taso seitsemän, josta on kuvakaappaus alla (Kuva 19), on huomattavasti kestoltaan pitempi kuin edelliset tasot ja sisältää useita uusia elementtejä. Tason alussa laserit ilmoittavat sijainneistaan näytöllä läpinäkyvillä punaisilla viivoilla ja pari mustaa palloa lähestyy pelaajahahmon takaa liikkuen tämän eteen. Tämän jälkeen laserit aktivoituvat ja laserviivat värjäytyvät vahvasti punaisiksi ja lisää mustia palloja alkaa ilmestyä pelaajan takaa.



KUVA 19. Kuvakaappaus tasosta 7. Punaiset viivat ovat lasereita. Pelaajahahmo on valkoinen pallo. Mustat punareunaiset pallot ovat pallosalamoita.

Taso kahdeksan on helpompi kuin taso seitsemän, siinä aikaisemmin luotuja pelikomponentteja on sijoitettu eri tavoin kuin aikaisemmissa pelikentissä. Tämä luo peliin uusia haasteita.

Voiton jälkeen peli siirtyy vaihtoehtoiseen aloitusikkunaan jonka voi nähdä alla (Kuva 20), siihen on lisätty värikäs taustakuva ja taustalla soi nyt rauhallinen musiikki. Tässä vaiheessa pelin esituotantovaihe on nyt valmis ja siirrymme tuotantovaiheeseen.



KUVA 20. Aloitusikkuna voiton jälkeen.

5.5. Inkrementti 4

Pelin viimeisimpään inkrementtiin, jossa on siirrytty tuotantovaiheeseen, lisäsin vielä tasot yhdeksästä yhteentoista, näitä varten toteutin kaksi uutta käskysarjaa ja tein vaihtoehtoiset versiot kahdesta aiemmin tehdystä pelikomponentista.

Lisätyt Blackballspawner2 ja Blackballspawner3 eroavat alkuperäisestä käskysarjasta uudelleen toistuvien toimintojen osalta (Kuva 21), niissä ei ole Cycle funktioita vaan ne toimivat pelkän Update funktion sisällä.

```
if (active){
    //Targeted
        if (timer < 0) { Spawner(playerpos.transform.position.x); }
    //Randomised
        print("Random spawn");
        if (step == 0) { Spawner(Random.Range(-19.5f, 19.5f)); step++; }
        if (step == 1) { Spawner(Random.Range(-19.5f, 0f)); step++; }
        if (step == 2) { Spawner(Random.Range(0f, 19.5f)); step++; }
        if (step == 3) { Spawner(Random.Range(-10f, 10f)); step++; }
        if (step >= 4 & timer < 0) { step = 0; }
    //Lines
        if (timer2 < 0) { Spawner(-17f); Spawner(17f); Spawner(0f); }
    }
```

KUVA 21. Käskysarja 10. Blackballspawner 2 if(active) funktio

Pelikomponenttien koon muuttamista varten loin uuden Spawner käskysarjan, joka eroaa aikaisemmista (Kuva 15, käskysarja 8) siinä, että se muuttaa luomansa pelikomponentin kokoa satunnaiskertoimen 0.1-0.8 mukaisesti. Tämä Spawner käskysarjan muutos vaati sen, että lisäsi pelin muistiin myös toisen blackball komponentin. Tämä lisäys tarvittiin, koska yksinomaan muistissa olevan komponentin arvojen muuttaminen olisi vaikuttanut sen kaikkiin kopioihin ei toivotusti myös muissa pelikentissä.

Tasossa 11 uudelle käytettiin myös Blackballspawner3 käskysarjaa mutta sen käyttämää pelikomponenttia vaihdettiin ja tähän uuteen komponenttiin lisättiin alla (Kuva 22) oleva käskysarja. Tämän käskysarja tehtävä on muuttaa pelikomponentin sijaintia sen pelikentälle kopioinnin jälkeen lisäämällä sen sijainnin x ja y koordinaattiin tietyille väleille osuvat satunnaisarvot. Sama komponentti myös liikkuu edestakaisin MoveRightLeft.cs käskysarjan avulla ja myös sen aloitusliikesuunta on satunnaistettu.

```
void Start () {
        this.transform.position = new Vector2(this.transform.position.x+
            Random.Range(-10f, 10f),
            this.transform.position.y + Random.Range(-10f, 5f));
        if (Random.Range(0f,1f) < 0.5f) {
        GetComponent<MoveRightLeft>().changedirection = true; }
        else { GetComponent<MoveRightLeft>().changedirection = false; }
    }
```

KUVA 22. Käskysarja 12. ChangeStartingPosition.cs Start() funktio

6 POHDINTA JA JOHTOPÄÄTÖKSET

Työn pohjalta voidaan samalla ohjelmointimenetelmällä jatko kehittää esim. kännyköihin tai muihin pelilaitteisin soveltuva Blackdrop peli tai vastaava lyhyt sovellus (apps). Työ toimii apuna oman peli-idean prototyypin esittelemisessä ja konseptin jatkotyöstössä. Peliä voisi vielä jatko kehittää monin eri tavoin. Esimerkiksi lisäämällä animaatiota, visuaalisia ilmauksia, vihollisälyä ja pelikenttien monimuotoisuutta.

Unity ohjelmointiympäristö on aloittelijaystävällinen ja helppokäyttöinen, internetistä voi helposti löytää ohjeita omiin tarpeisiinsa ja Unity yhtiön ylläpitämä ”asset store” palvelu on hyvä lähde valmiille pelikomponenteille sekä maksullisille että ilmaisille. Tämä ympäristö on hyvä työkalu jokaisen videopelejä kehittävän työkalupakkiin.

Peliteollisuus on nopeasti kasvava markkina, joka tuotti 30,4 miljardia dollaria vuonna 2016 ja joka työllistää laajasti koko informaatioalaa mm. käyttämällä sen tuottamia palveluita ja ohjelmointityökaluja. Videopeliteollisuus liittyy monien eri yhteyksien kautta kansantalouteen ja sen vaikutus on siten suurempi kuin vain sen suorat rahatulot. (Siwek. S.E 2017).

Videopeli teollisuus on taloudellisesti merkittävä toimiala myös Suomessa. Alan suurimmat toimijat ovat pelikonsoleita valmistavat yritykset, videopelejä kehittävät pelitalot ja niitä myyvät jälleenmyyjät. Suomessa merkittävimmät pelialan toimijat ovat Supercell (http://supercell.com/en/) ja Rovio (http://www.rovio.com/fi), sekä joukko kasvamassa olevia pienempiä yhtiöitä ja yrityksiä. Tästä syystä valitsin päättötyöni aiheeksi pelin ohjelmoinnin, koska olen kiinnostunut pelialasta ammattina.

7 LÄHTEET

aleksandr. 2004. Documentation, Unity scripting languages and you. Luettu 20.6.2017. https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

Charger Games. 2015. Unity - 2D Infinite Scrolling Background the Simplest Way. Katsottu 5.6.2017. https://www.youtube.com/watch?v=HrDxnMI7pCc

djgriffin. 2012. Video Game 7.wav. Ladattu 11.7.2017. https://freesound.org/people/djgriffin/sounds/172561/.

duck. 2010. Answer. Luettu 8.6.2017. https://unity3d.com/learn/tutorials/projects/survival-shooter/player-health.

ERH. 2007. string 1 loop.wav. Ladattu 11.7.2017. https://freesound.org/people/ERH/sounds/30192/.

Erokia. 2013. Elementay Wave 11.wav. Ladattu 11.7.2017. https://freesound.org/people/Erokia/sounds/183881/.

Francois Dominic Laramee. 1999. The Game Design Process. Luettu 9.10.2017. https://www.gamedev.net/articles/game-design/game-design-and-theory/the-game-design-process-r273/.

frankum. 2014. electronic music loop 001. Ladattu 11.7.2017. https://freesound.org/people/frankum/sounds/255585/.

GabrielAraujo. 2014. Powerup/success.wav. Ladattu 11.7.2017. https://freesound.org/people/GabrielAraujo/sounds/242501/.

gmachine88. 2009. Collision VS Trigger. Luettu 7.6.2017. https://forum.unity3d.com/threads/collision-vs-trigger.30174/.

Hasan, N. 2012. History of C# Programming. Luettu 20.6.2017. http://aboutcsharpprogramming.blogspot.fi/2012/09/history-of-c-programming.html.

Jason Schreier. 2017.The Story Behind *Mass Effect: Andromeda*'s Troubled Five-Year Development. Luettu 4.10.2017. https://kotaku.com/the-story-behind-mass-effect-andromedas-troubled-five-1795886428.

KeithK. 2011. Answer. Luettu 7.6.2017. http://answers.unity3d.com/questions/56947/setting-gameobject-as-target-in-another-script.html.

Kovac, J. 2007. C#/.NET history lesson. Luettu 16.6.2017. http://jameskovacs.com/2007/09/07/cnet-history-lesson/.

LeftyRighty. 2016. Post. Luettu 13.6.2017. https://forum.unity3d.com/threads/fading-in-out-gui-text-with-c-solved.380822/.

lineupthesky. 2015. Post. Luettu 9.6.2017. https://forum.unity3d.com/threads/c-how-to-use-time-deltatime-to-move-object-smoothly.307197/.

Lucas Meijer 1. 2010. Answer. Luettu 5.6.2017. http://answers.unity3d.com/questions/9675/is-unity-engine-written-in-monoc-or-c.html

Mrthenoronha. 2016. Space Game Theme Loop.wav. Ladattu 11.7.2017. https://freesound.org/people/Mrthenoronha/sounds/371516/.

Nick Pettit. 2015. How to Make a Loading Screen in Unity. Luettu 9.6.2017. http://blog.teamtreehouse.com/make-loading-screen-unity.

orangefreesounds. 2014. Magic Bells Musci Loop. Ladattu 11.7.2017. https://freesound.org/people/orangefreesounds/sounds/242082/.

Oskari Aalto. 2015. Mobile game product developement models. Luettu 4.10.2017. http://urn.fi/URN:NBN:fi:amk-2015120319123

percypersimmon. 2012. Wah Wah Wah Wah. Ladattu 11.7.2017. https://freesound.org/people/percypersimmon/sounds/156592/.

PiskelApp. 2017. Käytetty 8.6.2017. http://www.piskelapp.com/.

Player Health. 2017. Luettu 6.6.2017. https://unity3d.com/learn/tutorials/projects/survival-shooter/player-health.

Ralph Edwards. 2006. The game production pipeline: Concept to completion. Luettu 4.10.2017. http://www.ign.com/articles/2006/03/16/the-game-production-pipeline-concept-to-completion?page=1.

Stephen E. Siwek. 2017. Video games in the 21st century. Luettu 4.10.2017. http://www.theesa.com/wp-content/uploads/2017/02/ESA_EconomicImpactReport_Design_V3.pdf

Twisted Wave Online. 2017. Käytetty 8.6.2017. https://twistedwave.com/online/.

Unity3d. 2017. Build once, deploy everywhere. Luettu 11.10.2017. https://unity3d.com/unity/features/multiplatform

UnityScript versus Javascript. 2017. Luettu 5.6.2017. http://wiki.unity3d.com/index.php/UnityScript_versus_JavaScript

vdek. 2012. Post. Luettu 3.7.2017. https://forum.unity3d.com/threads/laser-beam-script-in-c.122500/.