Tampereen ammattikorkeakoulu

Unity 2D- PC-pelin siirtäminen mobiilialustalle

Aleksi Asikainen

Ville Raunio

OPINNÄYTETYÖ
Marraskuu 2022

Tietojenkäsittelyn tutkinto-ohjelma
Game Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Game Production

ASIKAINEN, ALEKSI & RAUNIO, VILLE:
Unity 2D- PC-pelin siirtäminen mobiilialustalle

Opinnäytetyö 39 sivua, joista liitteitä 0 sivua
Marraskuu 2022

Tämän opinnäytetyön tavoitteena oli käydä läpi eri vaiheita ja tekniikoita, joita liittyy tietokonepelin siirtämisessä mobiilille. Työssä käsitellään eri keinoja, joilla pelin alkuperäisen ”näppäimistö ja hiiri” -tyylisen pelattavuuden sai muutettua kosketusnäytöllä pelattavaksi peliksi. Työn tarkoituksena oli saada enemmän kokemusta mobiilipelien luomisesta Unity-pelimoottorilla ja samalla myös oppia mitä asioita on todella tärkeä ottaa huomioon mobiilipeliä valmistaessa.

Työn aikana yksi suurimmista haasteista oli pelin yleisen pelattavuuden säilyttäminen mahdollisimman saman tuntuisena kuin mitä peli tietokoneellakin oli. Pyrkimyksenä oli peliä siirtäessä säilyttää sen alkuperäinen tunne, jotta pelin tietokone- ja mobiiliversiota verratessa toinen versio ei saisi suurempaa huomiota.

Opinnäytetyön tuloksena peli saatiin siirrettyä onnistuneesti tietokoneelta mobiilille. Työssä selätettiin suurempien mobiilipelien valmistamisen haasteet ja huomioitiin erilaiset tekniikat, joilla pelistä sai mobiililla pelatessa mahdollisimman sulavan ja toimivan. Työtä tuki erityisen hyvin Unity-pelimoottorin dokumentaatio, josta oli suurin hyöty työn aikana.

Pelin siirtäminen tarjosi hyvän mahdollisuuden oppia mobiilipelien valmistamisesta. Työssä panostettiin osa-alueisiin, joilla on suuri vaikutus mobiilipelien pelattavuuteen.

Asiasanat: mobiilipeli, siirtäminen, unity-pelimoottori

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor of Business Administration
Game Production

ASIKAINEN, ALEKSI & RAUNIO, VILLE:
Unity 2D- PC-Game Porting to Mobile Platform

Bachelor's thesis 39 pages, appendices 0 pages
November 2022

The goal of this Bachelor's thesis was to go through the different steps and techniques involved in porting a computer game to mobile. The work covers different solutions used to transform the original "keyboard and mouse" style gameplay into a touch screen friendly experience. The purpose of the work was to gain more experience in mobile game development with the Unity-game engine and at the same time also learn what things are important to consider when making a mobile game.

During the work, one of the main challenges was to keep the overall playability of the game as similar as it was on the computer. During the porting process, the aim was to preserve the original feel of the game, so that there would be no unfair factors to consider when comparing the computer and mobile versions of the game.

As a result of the thesis, the game was successfully ported from the computer platform to the mobile platform. The work was particularly well supported by the Unity-game engine documentation, which was the greatest source of information during the work.

Key words: mobile game, porting, unity-game engine

SISÄLLYS

1 JOHDANTO 7
2 ALUSTA 8
2.1 Mobiililaitteet alustana peleille 8
2.2 Unity-pelimoottorin rooli modernien mobiilipelien tuottamisessa 9
3 GRAFIIKAN VISUAALINEN SUUNNITTELU 10
3.1 Unity-pelimoottorin grafiikka 10
3.2 Peliprojektin grafiikka mobiililla 10
4 TEKNISEN TOTEUTUKSEN SUUNNITTELU 13
4.1 Pelaajan ohjaus mobiilialustalla 13
4.2 Pelin vaikeustason hienosäätö 13
4.3 Muistinhallinnan tärkeys mobiilialustalla 14
4.4 Fysiikan käyttäminen peliobjekteissa 14
4.5 Unity Profilerin hyödyntäminen suorituskyvyn optimoinnissa 14
4.6 Koodin rakenteen optimointi 15
5 VISUAALINEN TOTEUTUS 16
5.1 Käyttöliittymän rakentaminen 16
5.2 Grafiikan asetuksien määritys mobiilille 23
5.3 Pelin valaistuksen muuttaminen mobiilille sopivaksi 24
6 TEKNINEN TOTEUTUS 26
6.1 Unityn valmistelut käyttäjän syötteen lukemiseen 26
6.2 Käyttäjän syötteen lukeminen 27
6.3 Pelaajan liikuttaminen 28
6.4 Vaikeustason ja hauskuuden lisäys 28
6.5 Pelaajan tähtäyksen toteuttaminen 28
6.6 Pelaajan ampuminen 30
6.7 Koodin rakenteen muutokset 31
6.8 Muistinhallinta 32
6.9 Unity Profiler-analysointi 32
7 PROJEKTIN VIIMEISTELY 35
8 POHDINTA 37
LÄHTEET 38

LYHENTEET JA TERMIT

Android	Käyttöjärjestelmä mobiililaitteille.
APK	Tiedostomuoto Android käyttöjärjestelmälle, jolla mahdollistetaan sovelluksien asennus.
Atan2	Matemaattinen funktio, joka palauttaa x- ja y-koordinaattien arvojen arkustangentin.
Canvas	Unity-pelimoottorin osa, johon käyttöliittymä rakennetaan.
Fysiikkamoottori	Simuloi fyysisten esineiden vuorovaikutusta muihin esineisiin. Peleissä esimerkiksi törmäykset vaativat yleensä fysiikkamoottoria.
HUD	Heads-up display. Pelin ruudulla oleva objekti, joka antaa tietoa pelaajalle.
Koontiversio	Projektin lähdekoodi muunnettuna omaksi sovellukseksi.
Kovakoodattu	Arvo, joka on syötetty koodiin suoraan. Se ei ole haettavissa tai muutettavissa.
Käyttöliittymä	Pelin osa, joka antaa pelaajalle tarvittavan tiedon ja hallittavuuden pelaamiseen.
Liitännäinen	Tietokoneohjelma, joka voidaan lisätä toiseen ohjelmaan sen toimintojen lisäämiseksi.
Luokka	Koodikielessä luokkaa voidaan kuvailla säiliöksi muuttujille ja toiminnoille. Sitä käytetään koodin organisointiin.
Object Pooling	Suunnittelumalli, jossa objekteja käytetään uudelleen tuhoamatta ja uudelleenluomatta niitä.
Objekti	Sisältää joukon loogisesti toiseensa liittyvää tietoa ja sen toiminnallisuutta.
Radiaani	Tasokulman suuruuden mittayksikkö.
Rect Transform	Transformia vastaava komponentti käyttöliittymän objekteille.
Scene	Unity editorissa oleva pohja, johon peli tai pelin taso luodaan.

Siirtäminen	Pelin siirtäminen alustalta toiselle.
Skripti	Kooditiedosto, joka sisältää yksittäiseen peliobjektiin liittyviä toiminnallisuuksia.
Tagi	
Tekstuuri	Osa objektia pelissä, joka antaa sille visuaalisen pinnan.
Transform	Peliobjektin data, joka sisältää sijainnin, koon ja rotaation.
Unity	Alustariippumaton pelimoottori.
Unity Profiler	Unity-pelimoottorin sisäänrakennettu työkalu pelin suorituskyvyn analysoimista varten.
Vakiomuuttuja	Muuttuja, jonka arvo ei voi muuttua sen määrittelyn jälkeen.
Vector2	Muuttuja, joka sisältää kaksi arvoa. X- ja y-koordinaatit 2D-muodossa.
Virtuaalinen ohjaussauva	Peliohjaimesta löytyvän sauvan vastaava versio virtuaalisena.

1 JOHDANTO

Mobiilipelien suosio nykypäivänä on todella korkealla tasolla. Tämän saman ovat myös todenneet monet suuremmat studiot pelialalla, joka on johtanut konsoli- ja tietokonepelien julkaisemiseen myös mobiilialustalla. PC-pelin siirtäminen mobiilille sisältää monia erilaisia tekijöitä, joista suurimpiin liittyy pelin suorituskyky sekä kuinka hyvin ohjaukset ovat mobiililla toteutettu.

Valitsimme siirrettäväksi peliksi aiemmin valmistamamme pelin nimeltä Mecha Hell. Kyseisen pelin mekaniikat olivat helppo muuttaa mobiilille käytettäväksi ja peli ei PC-alustallakaan vaatinut laitteelta paljoa, jotta se pyöri ja oli pelattavassa kunnossa.

Tässä työssä käydään läpi eri käytäntöjä ja osa-alueita, joita liittyy PC-pelin siirtämisessä mobiilille. Käytännöistä käydään läpi eri asioita, joita täytyy ottaa huomioon peliä siirtäessä mobiilille ja asioista, joihin täytyy kiinnittää huomiota, jotta mobiililla pelattavasta pelistä tulee mahdollisimman mobiiliystävällinen.

Pelin siirtämisen osa-alueiden läpikäyminen aloitetaan pelin grafiikasta, jossa suurin keskittymisen aihe on käyttöliittymän grafiikka ja sen muuttaminen mobiililaitteelle sopivaksi. Grafiikan jälkeen keskitytään pelin koodiin, jossa käydään läpi tarkemmin mobiilille valmistetun käyttöliittymän toimintaan saaminen. Koodissa käydään myös läpi eri keinoja pelin suorituskyvyn parantamiseksi sekä sen analysoimisen toimenpiteet.

2 ALUSTA

2.1 Mobiililaitteet alustana peleille

Mobiililaitteet ovat suuri alusta pelien julkaisulle. Manningin (2017, 1.) mukaan vuosi 2008 oli suurimpia harppauksia mobiilialustoille Applen iPhonen ja Googlen Android alustojen julkaisujen ansiosta. Shekarin (2017, 7.) mukaan vuona 2014 Android älypuhelimia oli myyty maailmalla noin 1 miljardia kappaletta. Mobiililaitteiden eri alustat, kuten Androidin Google Play Store ja iOS:n iTunes App Store tarjoavat pelien valmistajille loistavan mahdollisuuden tavoittaa ja saada pelit pelattavaksi näiden laitteiden suurille käyttäjäkunnille (Sinicki 2017, 1.) Pelintekijöille ja mobiililaitteilla pelaaville pelin siirtäminen on todella houkutteleva idea. Pelintekijät saavat suuremman määrän pelaajia kerrytettyä pelille ja pelaajat taas pääsevät pelaamaan heille entuudestaan tuttua ja viihdyttävää peliä missä vain he haluavat.

Kun pelejä valmistetaan mobiilille, niin odotukset peliin sen pelaajilta ovat huomattavasti pienemmät. Tämä tarkoittaa myös sitä, että aika ja resurssit pelin tekemiseen ovat pienemmät verrattuna tietokone- ja konsolipeleihin (Sinicki, 2017, 1.) Sinicki (2017, 1.) vertaa myös hyvin konsolipelien valmistamista mobiilipelien valmistamiseen siten, että konsolialustalle valmistaessa peliä se tulee kilpailemaan suurimpien pelien kanssa.

Manningin (2017, 1.) mukaan mobiilipelit nykypäivänä ovat joko yksinkertaisia pelejä, enemmän komplekseja pelejä tai siirtoja aiemmin julkaistusta pelistä. Näistä ehdottomasti yksinkertaisemmat pelit hallitsevat mobiilipelien markkinaa, mutta nykyään pelinkehittäjät ovat kyenneet julkaisemaan mobiilille pelejä mitä ei ollut teknisesti mahdollista aiemmin julkaista. Mobiililaitteiden suorituskyky on viimeisien vuosien aikana noussut todella paljon, jonka ansiosta laajempien ja suorituskyvyltään vaativien pelien valmistaminen ei ole nykyään yhtä suuri haaste.

2.2 Unity-pelimoottorin rooli modernien mobiilipelien tuottamisessa

Mobiilipeliä valmistaessa on tärkeää, että pelimoottori on kykenevä täyttämään mobiilialustan tarpeet. Nämä kriteerit Unity-pelimoottori täyttää. Shekar (2017, 17) määrittelee Unityn ekosysteemiksi, joka sisältää laajan määrän erilaisia mekaniikkoja ja työkaluja 2D ja 3D pelien valmistamiseen.

Unityn statistiikat modernien mobiilipelien suhteen ovat korkealla tasolla. 1000 suosituimmista puhelinpeleistä on valmistettu Unityn avulla ja yli 50 % uusista mobiilipeleistä valmistetaan Unityn avulla (Unity 2022).

Unity-pelimoottorin alustariippumattomuuden ansiosta aiemmat Unity projektit ovat teknisesti katsoen erityisen vaivattomia siirtää miltä vain alustalta mobiilille. Pelimoottorin sisäänrakennetut ominaisuudet mobiilipelin toteutusta varten helpottavat myös siirtämisprosessia erityisen paljon.

Kuten Finnegan (2015, 1.) kuvailee Unity pelin tuottamista, voit aloittaa luomalla ilmaislisenssillä tietokonealustalle pelin ja myöhemmin napin painalluksella siirtää pelin mobiilialustalle. Unity-pelimoottorin yksi vaikuttavimmista ominaisuuksista on pelin siirtämisen helppous. Napin painalluksella peli voi pyöriä millä alustalla tahansa. Pelattavuudelta esimerkiksi ohjauksien luominen uudelle alustalle jää itse pelintekijän harteille, mutta pelin ensimmäistä kertaa käynnistäminen uudella alustalla on suurimmaksi osaksi pelintekijöitä eniten vaivaava prosessi.

3 GRAFIIKAN VISUAALINEN SUUNNITTELU

3.1 Unity-pelimoottorin grafiikka

Unity tarjoaa useita eri asetuksia, joita voidaan säätää yksittäiselle pelin grafiikalle tai pelin kokonaiselle ulkonäölle. Pelin suorituskykyyn vaikuttaa Unity-pelimoottorissa myös todella paljon Render Pipeline, joka projektille valitaan. Render Pipeline toteuttaa erilaiset käytännöt, joiden avulla otetaan Scenen sisältö ja näytetään se näytöllä (Unity Documentation 2022g.)

Unity-pelimoottorin valmiiksi rakennettuihin Render Pipelineihin sisältyy Universal Render Pipeline, jota kyseisessä projektissa käytettiin. Suorituskyvyltään Universal Render Pipeline on Unityssä yksi parhaimmista vaihtoehdoista puhelinpelin valmistuksessa sen skaalautuvuuden ansiosta. Kuten Unityn omassa dokumentaatiossa kuvaillaan, Universal Render Pipeline tarjoaa artisteille käyttäjäystävällisiä työtapoja, joiden avulla saa helposti luotua optimoituja grafiikoita monille eri alustoille, kuten mobiililaitteille, konsoleille ja tietokoneille (Unity Documentation 2022i.)

Mobiilipeliä tuottaessa täytyy kiinnittää suurempi määrä huomiota pelin ladattavaan kokoon kuin mitä tietokoneella täytyy. Pelin tallennettavaan kokoon vaikuttaa paljon tekstuurien koko.

3.2 Peliprojektin grafiikka mobiililla

Käyttöliittymän uudelleen suunnittelu mobiilille sisältää paljon eri tekijöitä, joihin täytyy kiinnittää huomiota. Ruudun koko pienenee huomattavasti, joten erityisesti tekstin ja ikonien luettavuuteen on erittäin tärkeä panostaa.

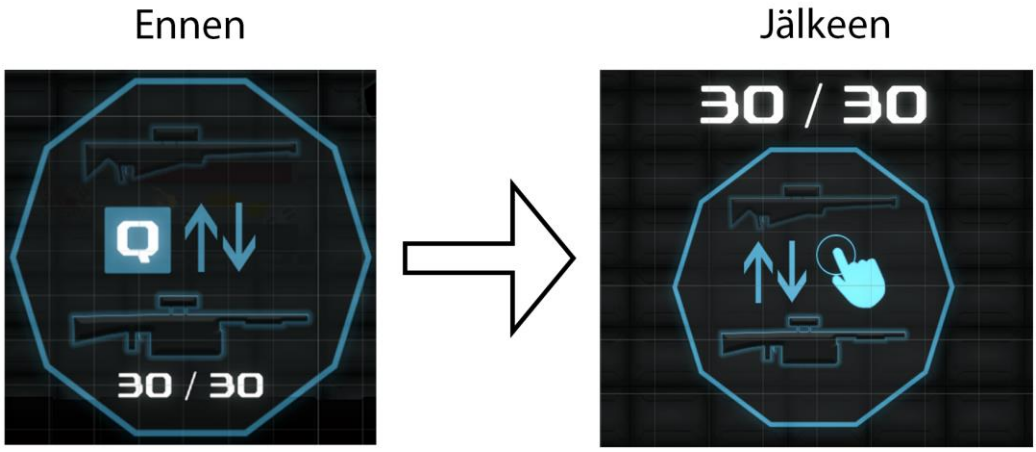
Kaikki pelin ohjaukset tietokoneversiossa toteutui näppäimistöllä ja hiirellä, joten käyttöliittymässä oli paljon tilaa, eikä pelissä ollut tarvetta asettaa ruudulle klikattavia nappeja.

Kun mobiilipelin ohjaukset tehdään kosketusnäyttö ensimmäisenä prioriteetilla, niin täytyy saada ruudulle mahtumaan useampi eri objekti. Jotta pelaaja pystyy toimivasti olemaan vuorovaikutuksessa peliin, täytyy käyttöliittymään lisätä uusia vuorovaikutukseen mahdollistavia objekteja pelaajalle. Nämä tulevat yleensä ruudulla olevien ohjaussauvojen ja nappien muodossa.

Käyttäjän syötteen grafiikoihin tavoiteltiin tietokoneversiossa mahdollisimman ristiriidaton ja simppeli tyyli. Tämä sama tyyli pyrittiin säilyttämään myös pelin mobiiliversiossa. Ristiriidattomat ja simppelit ikonit pelin tietokoneversiossa antoivat myös hyvän mahdollisuuden uudelleen käyttää niitä mobiiliversiossa erilaisilla keinoilla.

Suuren osan käyttöliittymän grafiikoista sai uudelleen käytettyä mobiilille muodostamalla ne ruudulla oleviksi napeiksi ja tekemällä grafiikkaan muutoksia. Esimerkkinä tästä voi käyttää käyttöliittymässä olevaa objektia, joka näyttää käytössä olevan aseen ja luotimäärän.

Kaikki muu aiemmin valmistettu grafiikka siirtyi saumattomasti mobiilille. Pääosin projektin suurin keskittymisen aihe oli kosketusnäytöllä pelattavuuden mahdollistavat grafiikat. Kuten kuvasta 1 näkee, pienillä muutoksilla aiemmin valmistettu grafiikka saatiin myös hyvään käyttöön pelin mobiiliversiota varten.



KUVA 1. Pelin aseinfon muutos. Kuvassa näytetään pienet muutokset, joilla grafiikasta saatiin mobiilille käyttökelpoinen.

Käyttöliittymän uusiin grafiikkoihin sisältyi kriittiset osat pelattavuuden kannalta, joihin ei tietokoneversiossa ollut tarvetta. Nämä olivat virtuaalisen ohjaussauvan grafiikat ja pelin paussi – napin grafiikka. Virtuaalisen ohjaussauvan grafiikkoihin kuului ohjaussauva sekä sen tausta. Näiden käyttöliittymään laitettavien grafiikoiden avulla saa käyttäjän syöte tarvittavat lisät, jotka ovat liikkuminen, ampuminen ja pelin keskeyttäminen.

4 TEKNISEN TOTEUTUKSEN SUUNNITTELU

4.1 Pelaajan ohjaus mobiilialustalla

Käyttäjän syötteen kohdalla muuttaessa peli tietokoneelta puhelimelle sopivaksi tarvittiin muutoksia käyttöliittymään, virtuaalisten ohjaussauvojen toteuttamiseen ja koodin rakenteeseen.

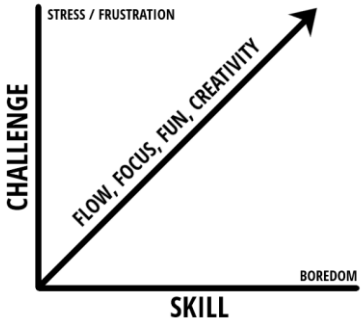
Muuttaessa pelaajan ohjaus näppäimistöstä puhelimen ruudulle, täytyi miettiä millä tavalla ohjaus puhelimelle kannattaa toteuttaa. Pelikonsolilla pelattaessa ylhäältäpäin kuvatuissa peleissä on tavallisesti kaksi ohjaus sauvaa. Vasemmalla ohjataan hahmoa ja oikealla tähdätään siihen suuntaan mihin halutaan ampua. Tietokoneversiossa liikkuminen tapahtui näppäimistöllä, joten pelin koodiin täytyi tehdä muutoksia muuttaessa se puhelimelle toimivaksi.

4.2 Pelin vaikeustason hienosäätö

Pelin työpöytä versiossa pelin vaikeustason säätämisessä oli ongelmana vaikeustason liian suuri nousu tietyssä pelin vaiheessa. Peliä testattaessa huomattiin pelissä tietyssä kohdassa vaikeustason nousu liian nopeasti. Tähän täytyi tehdä muutos, jotta pelaajalla pysyisi mielenkiinto peliin pidemmän aikaa.

Sinun pitää aloittaa esittämään merkittäviä haasteita, jossain välissä peliä, koska muuten siitä tulee tylsää. Pelit ovat hauskoja niin kauan, kun me opimme. Jos esität aivoille liian vähän stimulointia tai haastetta, siitä tulee väsynyt. Me nopeasti etsimme jotain muuta tekemistä. (Sinicki, 2017, 10.)

Voimme nähdä kuvassa 2 miten pelin haastavuuden ja pelaajan taidon olisi hyvä kulkea samassa tahdissa. Tämän kaavan pohjalta voimme lähteä rakentamaan pelin vaikeustasoa sopivammaksi pelaajille. Meidän tulee välttää liian suuret vaikeustason nousut, mutta pitää vaikeustason nousua sopivana ja haastavana pelaajalle. Pelaajalle tulee esittää uusia vihollistyyppejä ja vihollisten hyökkäyksiä, jotta peli pysyy kiinnostavana.



KUVA 2. Vaikeustason sopivaa nousua kuvaava kaava.

4.3 Muistinhallinnan tärkeys mobiilialustalla

Object Pooling on tapa optimoida projektin suorituskykyä paremmaksi ja vähentää kuormaa mikä kohdistuu laitteen suorittimelle tilanteessa, jossa nopeasti luodaan ja poistetaan objekteja. Se on hyvä suunnittelumalli pitää mielessä suorittimen prosessointinopeuden säästämiseksi tärkeämpiä tehtäviä varten. Object Pool voi käyttää esimerkiksi koodimaailmasta tuttua listaa objektien instanssien säilyttämiseen ja uudelleenkäyttöön. (Unity Documentation 2022g).

4.4 Fysiikan käyttäminen peliobjekteissa

Itsestään selvin tapa parantaa toimintojen suorituskykyä on välttää niiden käyttämistä niin paljon, kuin mahdollista. Kaikille liikkuville objekteille meidän tulisi miettiä vaativatko ne fysiikkamoottoria toimiakseen. Jos ei, niin meidän tulisi etsiä tapoja, jotka eivät ole niin paljon resursseja vaativia ja korvata ne jollain yksinkertaisemmalla ja vähemmän resursseja vievällä. Chris Dickinson (2017, 1.)

Peliä suunnitellessa tulee miettiä, mitkä objektit tarvitsevat fyysisiä ominaisuuksia. Jos objekti on yksinkertainen ja se ei vaadi fyysisiä ominaisuuksia, voidaan sille koodin avulla luoda vaihtoehtoinen ratkaisu mikä ei vaadi fysiikkamoottorin käyttämistä.

4.5 Unity Profilerin hyödyntäminen suorituskyvyn optimoinnissa

Unity Profilerilla paikannetaan pelin suorituskyvystä löytyviä suuria kompastuskiviä. Unity Profilerin ja pelin ollessa päällä seurataan näkymään tulevia suuria piik-

kejä. Näistä näkymän piikeistä voidaan katsoa mikä pelin toiminnallisuus aiheuttaa suorituskyvyn hidastumiset. Asiaa voidaan katsoa siltä kannalta, että tuleeko piikki pelin valikossa vai ihan peliä pelattaessa. Pelissä tulleet piikit ovat kriittisempiä, koska ne voivat vaikuttaa pelaajan pelikokemukseen negatiivisesti. Kuvasta 3 näemme millä tavalla Unity Profiler näyttää pelissä tapahtuneet piikit suorituskyvyssä.

KUVA 3. Havainnollistaa graafisesti isot piikit näkymässä.

4.6 Koodin rakenteen optimointi

Koodia kirjoittaessa monesti tulee tehtyä luonnoksia tietynlaisiin toteutuksiin ja niiden testauksiin. Tällaisessa luonnostelussa tulee monesti kirjoitettua turhia rivejä koodiin.

Kun kirjoitat koodia, sinun täytyy pitää silmällä tulevaisuutta. Yhtenä päivänä olet päivittämässä tai korjaamassa peliäsi, sen pitäisi olla kivuton kokemus. Kuten on mainittu, niin vähemmän koodia tarkoittaa parempaa koodia. (Sinicki, 2017, 10.)

Käytä vakiomuuttujia arvoihin, joiden lukujen ei tarvitse vaihtua missään vaiheessa. Tällöin pelimoottori osaa varata muuttujille rajatun määrän muistia ja silloin ne eivät varaa turhaa muistia pelimoottorista. (Sinicki, 2017, 10.)

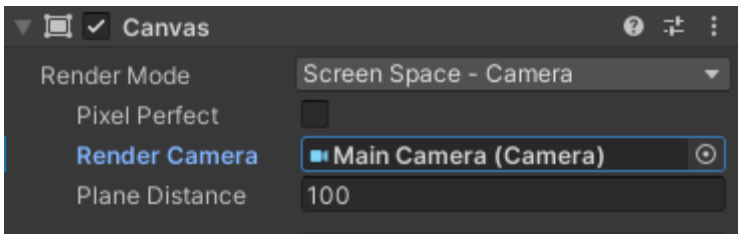
5 VISUAALINEN TOTEUTUS

5.1 Käyttöliittymän rakentaminen

Unity-pelimoottorin käyttöliittymän runkoa kutsutaan termillä Canvas. Canvas on alue, jonka sisällä kaikkien käyttöliittymän elementtien kuuluisi olla (Unity Documentation 2022b).

Unity projektiin lisätessä Canvaksen ensimmäistä kertaa, tulee projektin hierarkiaan myös uusi peliobjekti nimeltään Event System. Event System on ominaisuus, jonka avulla voidaan lähettää pelin eri objekteille tapahtumia eri syötteillä (Unity Documentation 2022d).

Kun Canvas peliobjekti lisätään projektiin, niin siihen on oletuksena liitettynä kolme eri komponenttia. Canvas komponentin avulla voidaan asettaa, että mihin käyttöliittymä renderöidään. Projektissamme renderöimme käyttöliittymän kulkemaan pelin kameran mukana. Tämän avulla käyttöliittymä kulkee koko ajan pelaajan kameran mukana ja toimii hyvin HUDina. Kuvasta 4 näkee kuinka Unityssä pelin käyttöliittymä saatiin liitettyä kameraan.

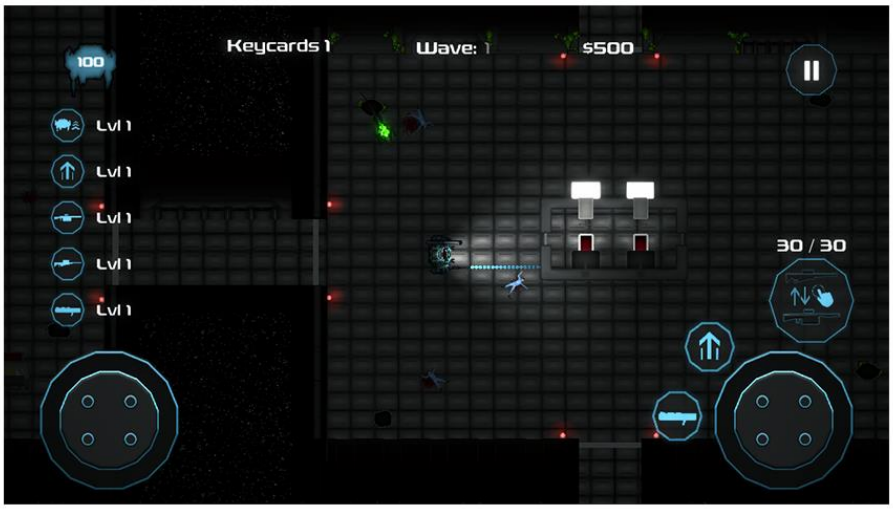


KUVA 4. Canvas komponentin ominaisuuksia. Kuvasssa näkyy, kun Canvas komponentin renderöinti moodi on asetettu kameralle ja pelin pääkamera on asetettu renderöinti kameraksi.

Canvas Scaler komponentin avulla saadaan käyttöliittymä skaalattua kolmella eri keinolla, jotka ovat pikseli koon mukaan, fyysisen koon mukaan tai näytön koon mukaan. Projektissamme määritimme käyttöliittymän skaalautumaan näytön koon mukaan, ja vielä tarkemmin näytön korkeuden mukaan. Canvas Scaler komponentille voidaan sen skaalautuessa näytön mukaan asettaa referenssi resoluutio. Referenssi resoluutioksi projektissamme laitoimme 1920-leveyden ja

1080-korkeuden. Näiden asetusten avulla käyttöliittymä skaalautuu hyvin, jos mobiililaitteessa on isompikin näyttö. Kuvasta 5 nähdään ero kuvasuhteiden välillä.

Kuvasuhde: 16:9



Kuvasuhde: 18:9

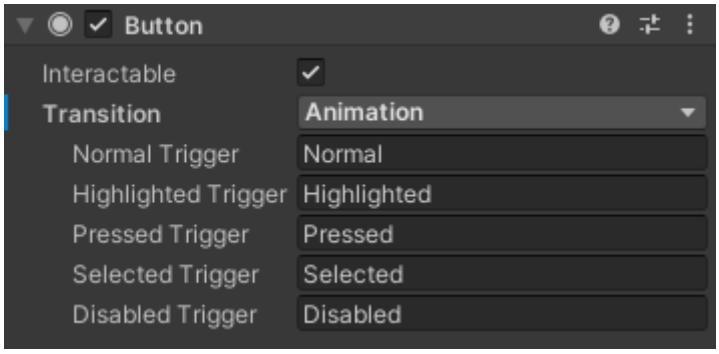
KUVA 5. Käyttöliittymä kahdella eri kuvasuhteella. Kuvasta näkee käyttöliittymän skaalautumisen kahden eri kuvasuhteen välillä.

Jotta käyttöliittymään saadaan lisättyä toiminnallisuutta, lisätään Canvas peliobjektiin Buttoneita.

Button elementti vastaa pelaajan painallukseen ja sillä saadaan käynnistettyä tai varmistettua toiminto (Unity Documentation 2022e).

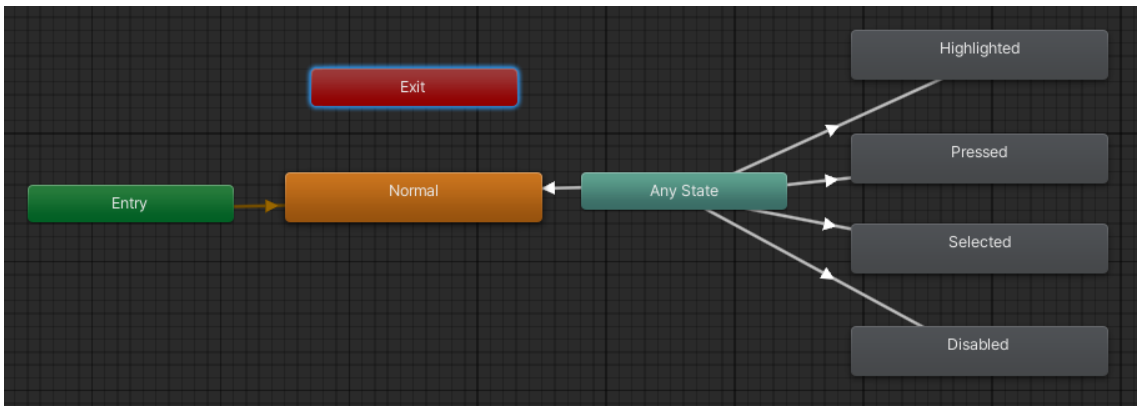
Unityn Button elementtiä käytimme projektissa luomaan näytöllä olevat napit, joiden avulla pelaaja saa käytettyä samoja mekaniikkoja kuin tietokoneversiossakin. Buttoneille saa myös valittua neljästä eri vaihtoehdosta, että mitä itse Buttonille tapahtuu visuaalisesti, kun sitä painetaan.

Pelissä oli kaksi nappia, jotka kaipasivat vielä enemmän responsiivisuutta. Jotta näihin kahteen nappiin saatiin responsiivisuutta, vaihdettiin niiden Transition asetus Animationiksi. Kuvassa 6 näytetään vielä tarkemmin eri animaatio tilat, joiden avulla animaatio toteutetaan, kun joku edellytyksistä siihen täyttyy.



KUVA 6. Button elementin Transition asetukset. Kuvasta näkee Button elementin asetukset, kun Transition on vaihdettu Animationiksi.

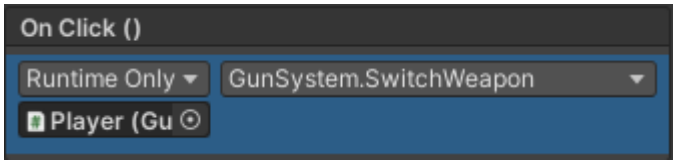
Button elementille saadaan samassa yhteydessä luotua Animator Controller. Animator Controllerin avulla saadaan hallittua animaatiopätkien toimintaa, jotka ovat hahmossa tai peliobjektissa (Unity Documentation 2022f). Buttonin automaattisesti generoidussa Animator Controllerissa tulee mukana myös tarvittavat animaatio klipit sekä siirtymät animaatiosta toiseen. Kuvassa 7 nähdään Button peliobjektin Animator Controller, jossa saadaan hallittua animaatioiden kulkua vielä tarkemmin.



KUVA 7. Buttonin Animator Controller. Kuvassa näkyy automaattisesti generoitu Animator Controller.

Buttonissa oleva ominaisuus On Click mahdollistaa skriptissä olevan metodin toteuttamisen, kun buttonia painetaan. On Click funktio on Unityssä oleva tapahtuma, jonka Unity herättää nappia painettaessa (Unity Documentation 2022e).

Buttonin On Click ominaisuuteen voidaan liittää peliobjekteja. Pelin aseenvaihtonappiin liitettiin pelaajahahmo. Pelaajahahmoon on liitettynä skripti, joka sisältää metodin aseen vaihtamista varten. Tämä metodi laitettiin toteutumaan buttonia painettaessa. Kuvasta 8 näkee kun Button peliobjektiin on liitetty haluttu skripti.



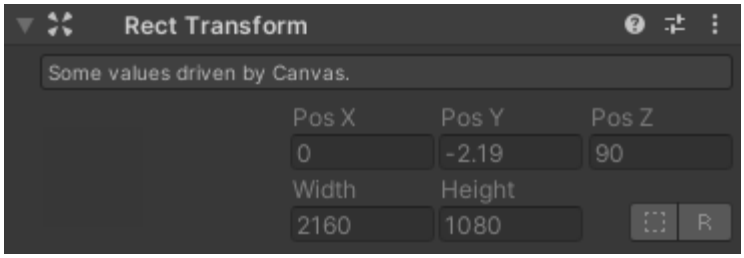
KUVA 8. Buttonin On Click ominaisuus. Kuvassa näkyy aseenvaihtonapin On Click ominaisuus, joka toteuttaa pelaajahahmo aseen vaihdon.

Canvas peliobjektin sisälle lisätään eri peliobjekteja, jotka toimivat käyttöliittymän osina. Näitä eri käyttöliittymän osia saa sijoiteltua käyttöliittymään haluttuun kohtaan muuttamalla niiden Rect Transformin Position X ja Position Y arvoja. Kuvasta 9 näkee miten Rect Transform ja Transform poikkeaa toisistaan.

KUVA 9. Rect Transform ja Transform. Kuvassa näkyy kuvankaappaukset Rect Transformista ja Transformista Unity Editorissa.

Käyttöliittymän objekteilla on Transformin sijaan Rect Transform. Rect Transform on 3D-peliobjektista löytyvän Transformin vastaava komponentti, mutta käyttöliittymän peliobjekteille (Takoordyal K, 2020, 4.2.)

Pelin Canvas peliobjektin skaalautuessa näytön resoluution mukaan, muuttuu Canvas objektin Width ja Height arvot resoluution mukaisesti. Kuvasuhteella 18:9, on Width arvo 2160 ja Height arvo 1080. Kuvasta 10 näkee miten Rect Transformin Width ja Height arvot menevät näytön koon mukaan.



KUVA 10. Canvas objektin Rect Transform. Kuvassa näkyy kuvankaappaus Canvasin Width ja Height arvoista, jotka muuttuvat näytön koon mukaan.

Käyttöliittymän objektien, esimerkiksi virtuaalisten ohjaussauvojen Rect Transformin arvoja tässä tapauksessa muutetaan katsoen näytön leveyttä ja korkeutta. Virtuaalinen ohjaussauva on täysin keskellä ruutua, jos sen Rect Transform kom-

ponentin X ja Y arvot ovat 0. Jos X arvon muuttaa negatiiviseen lukuun, niin ohjaussauva siirtyy näytön vasemmalle puolelle. Sauva siirtyy näytön oikealle puolelle X arvon ollessa positiivinen. Rect Transformin Y arvon ollessa positiivinen luku objekti siirtyy ylöspäin. Arvon ollessa negatiivinen luku, se siirtyy alaspäin.

Pelin ohjaussauvat asetettiin näytön oikeaan ja vasempaan alakulmaan muuttamalla Rect Transformin X arvot ampumissauvalle positiiviseksi luvuksi ja liikkumissauvalle negatiiviseksi luvuksi. Molempien ohjaussauvojen Rect Transformin Y arvo asetettiin samaksi negatiiviseksi luvuksi, jotta ne ovat korkeudella samalla tasolla.

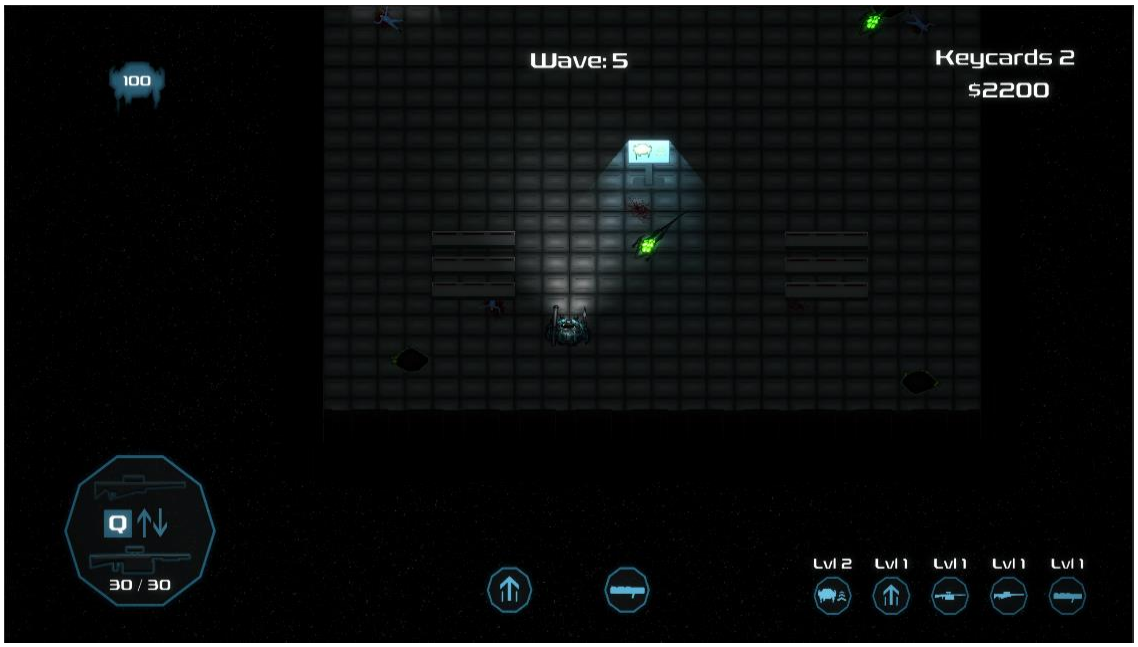
Pelaajan taitoikoneille ja aseinfolle valittiin uudet paikat virtuaalisten ohjaussauvojen tullessa pelin käyttöliittymään mukaan.

Robotin taitoikonit sijoitettiin robotin elintason alle vertikaalisessa suunnassa. Tämän muutoksen avulla pelaajan eri statukset löytyvät nyt yhdeltä puolelta näyttöä. Aseinfo uudelleen käytettiin aseenvaihto nappina, joka sijoitettiin näytön oikealle puolelle ammuntasauvan yläpuolelle.

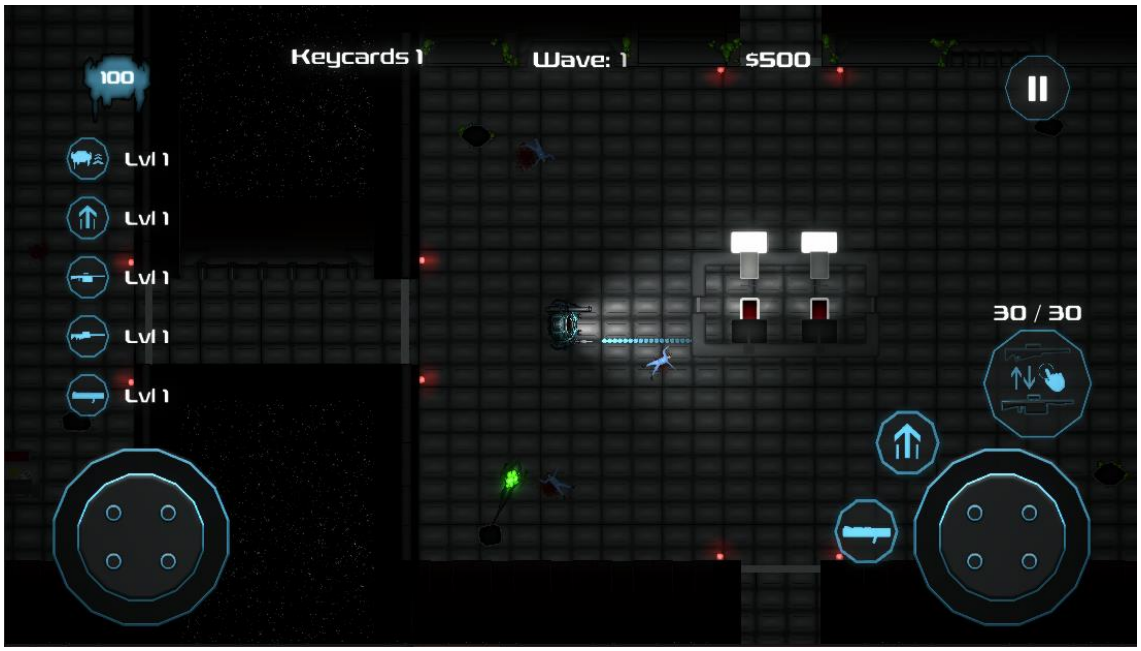
Peli tuotettiin puhelimelle niin, että sitä pelattaisiin pitämällä puhelinta vaakasuunnassa. Tällä tavalla pelin perspektiivi muistuttaisi mahdollisimman paljon sitä mitä se alun perin tietokoneella oli ja ohjaukset saisi myös mahdollisimman helppokäyttöisiksi. Kuvassa 11 nähdään miltä peli ja sen käyttöliittymä saatiin näyttämään puhelimella pelatessa.

KUVA 11. Pelin pelaaminen mobiililla. Kuvassa näkyy, kun peliä pelataan mobiililaitteella. Pelin kuvakulma on lukittu vaakasuuntaiseen perspektiiviin.

Pelin tietokoneversion käyttöliittymä verratessa mobiiliversioon, tyyli saatiin säilytettyä lähestulkoon samana. Ennen käyttöliittymän toteutusta, suurin pelko oli liian suuri objektien määrä ruudulla. Käyttöliittymään saatiin tehtyä pelattavuuteen vaikuttavia muutoksia, joilla käyttöliittymän tilaa säästyi hyvin. Kuvien 12 ja 13 vertailussa nähdään erot pelin tietokone- ja mobiiliversion käyttöliittymien välillä.



KUVA 12. Kuvankaappaus pelin tietokoneversiosta. Kuvasta näkee pelin tietokoneversion käyttöliittymän.



KUVA 13. Kuvankaappaus pelin mobiiliversiosta. Kuvassa näkyy mobiiliversion käyttöliittymään tehdyt lisäykset ja muutokset.

5.2 Grafiikan asetuksien määritys mobiilille

Yksittäisen tekstuurin asetuksia muuttaessa, asetettiin päälle asetus, jonka avulla voidaan tekstuurille asettaa omat asetukset, kun projekti toteutetaan Android alustalle. Asetuksista muutettiin tekstuurin maksimikokoa, koon muuttamisen algoritmia, formaattia ja kompressoinnin laatua.

Käyttöliittymän tekstuureilla suurimman eron positiivisesti tuotti tekstuurin formaatin muuttaminen. Tekstuuriasetuksien optimoinnilla pelin tallennettavaa kokoa saatiin optimoitua huomattavasti pienemmäksi ja samalla myös parannettua pelin suorituskykyä ilman suurempaa visuaalista eroa.

Esimerkkitapauksena optimoitavasta grafiikasta käytetään pelin virtuaalisen ohjaussauvan grafiikkaa. Tekstuurin ladattava koko ennen muutoksia oli 144 kilobittiä. Tekstuurin maksimikooksi muutettiin 256 pikseliä, joka pienensi tekstuurin koosta 384 x 384 kokoon 256 x 256. Tämä muutos aiheutti tekstuurille puhelimen ruudultakin nähtäviä muutoksia huonompaan suuntaan. Kompressoinnin laatu asetettiin parhaalle tasolle, jonka avulla pahimmat epäkohdat saatiin tekstuurista karsittua pois. Kyseisen tekstuurin tallennettava koko saatiin näillä keinoilla tiputuettua 6,9 kilobittiin. Kuten kuvasta 14 nähdään, visuaalinen ero asetuksien muuttamisen jälkeen ei ollut suuri.

KUVA 14. Vertailu tekstuurin ulkonäössä muutosten jälkeen.

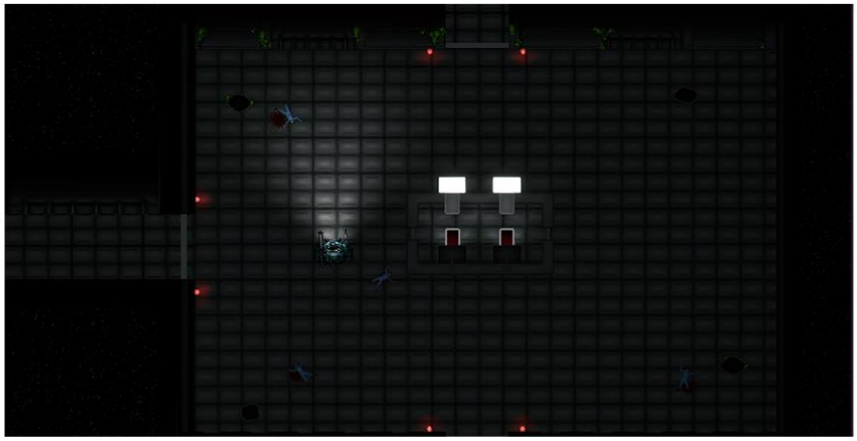
5.3 Pelin valaistuksen muuttaminen mobiilille sopivaksi

Pelin alkuperäinen valaistus tietokoneversiosta ei toiminut mobiililla, joten muutoksia täytyi tehdä. Pelin valaistukseen käytettiin Unityn sisäänrakennettua 2D-valaistus komponenttia. Komponentin avulla toteutettiin jokainen yksilöllinen valo sekä yleinen pimeys pelin maailmassa. Näistä valaistuksen osista mobiilille tehtiin muutoksia vain pelin yleiseen pimeyteen.

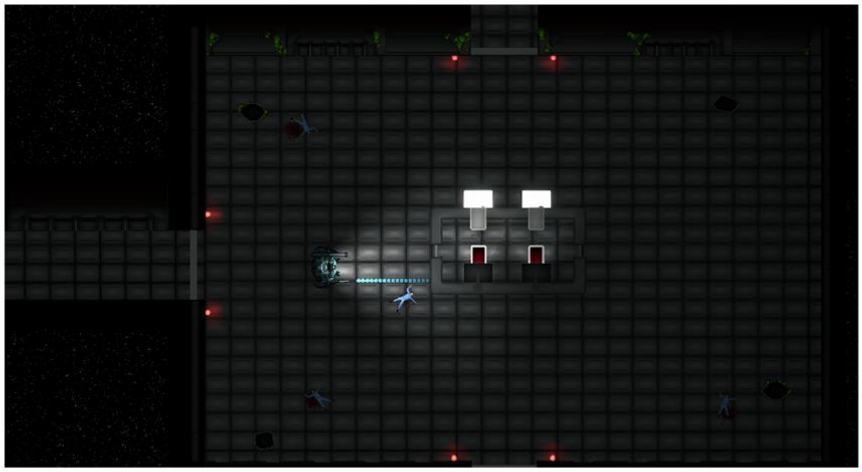
Global Light 2D- peliobjektin Intensity asetuksen avulla voidaan peliin muokata pelin yleisen valaistuksen määrää. Asetusta nostettiin pienellä määrällä ylöspäin, jotta pienemmälläkin näytön kirkkaudella pelin ympäristön erottaa hyvin.

Pelimaailman taustalla olevan tähtitaustan kirkkautta nostettiin myös, koska alkuperäinen kirkkaus ei riittänyt mobiiliversiolla. Pelimaailman taustalla olevat tähdet olivat myös yksi tekstuuri osa peliä, mutta tähän tekstuurin valaistus ei vaikuttanut. Lisää kirkkautta tähän tekstuuriin saatiin nostamalla sen värityksen arvoja. R, G ja B arvoja nostaessa suuremmaksi tekstuurin väritys muuttui vaaleammaksi, joka taas sai ulkonäön vaikuttamaan siltä, että tähtien kirkkautta on nostettu. Kuvasta 15 näkee erot tietokone- ja mobiiliversioiden valaistuksen välillä.

Tietokone



Mobiili

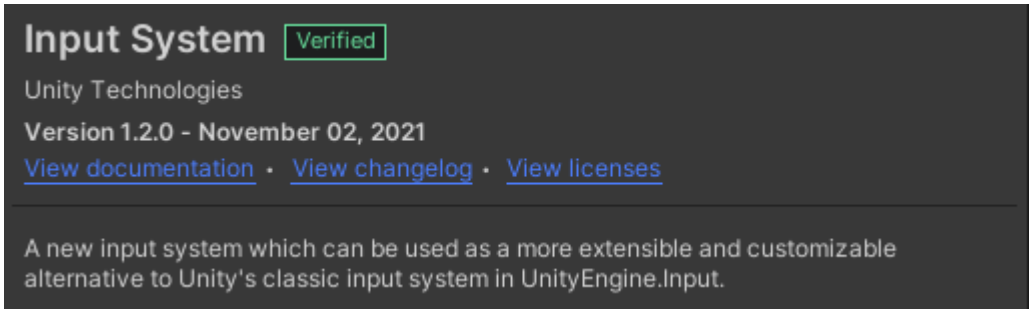


KUVA 15. Valaistuksen vertailu pelin versioiden välillä.

6 TEKNINEN TOTEUTUS

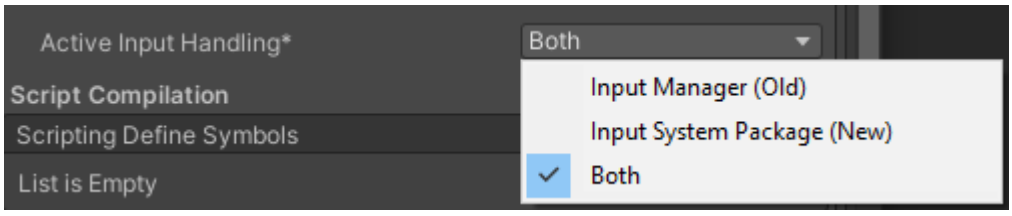
6.1 Unityn valmistelut käyttäjän syötteen lukemiseen

Virtuaaliset ohjaussauvat toteutettiin käyttäen Input System liitännäistä ja sen kirjastossa olevia lisäosia. Kuvassa 16 näkymä asennetusta lisäosasta Unity-projektiin.



KUVA 16. Unity Input System liitännäinen.

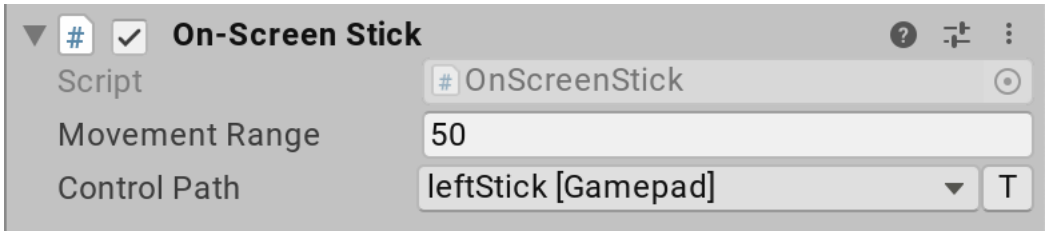
Asettaessa Unity käyttämään uutta Input System liitännäistä tulee projektin asetuksista käydä vaihtamassa Unity käyttämään uutta liitännäistä. Jotkin toiminnot jätettiin käyttämään vanhaa Input Manageria, joten projektissa valittiin Unity käyttämään molempia vanhaa ja uutta Input System liitännäistä. Kuvassa 17 valinta asetuksesta, jolla saa Unityn käyttämään vanhaa ja uutta Input System liitännäistä.



KUVA 17. Kuvankaappaus Unity-projektin asetuksista. Näiden projektin asetus säädöksien jälkeen projektin koodissa voidaan viitata molempien liitännäisten toimintoihin ja ne saadaan toimimaan halutulla tavalla.

6.2 Käyttäjän syötteen lukeminen

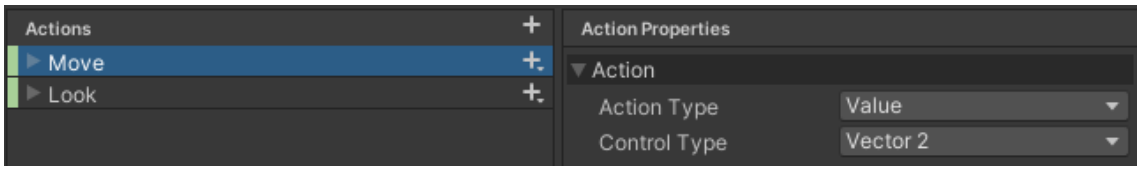
Käyttöliittymään luotuun ohjaussauvaan tuli liittää Input System liitännäisen mukana tullut skripti nimeltä On-Screen Stick. Sen avulla Input System pystyy kutsumaan tämän objektin metodeja. On-Screen Stick skripti mahdollistaa ohjaussauvan syötteestä saatavat arvot, joita käytetään pelaajan tähtäämiseen, liikuttamiseen ja ampumiseen. Kuvassa 18 On-Screen Stick skripti liitettynä ohjaussauvan peliobjektiin.



KUVA 18. Input System liitännäisen mukana tullut On-Screen Stick skripti.

Unityn uudessa Input Systemissä tulee määritellä millä ohjaimen tai näppäimistön painikkeilla käyttäjän syöte otetaan talteen. Tähän tarkoitukseen luodaan Input Action Asset tiedosto. Action asset on tiedosto, johon sisältyy käyttäjän syötteen toiminnot ja niitä käyttävät ohjaus profiilit. (Unity Documentation 2022c).

Tässä projektissa tarvitsimme molemmille ohjaussauvoille oman toiminnon. Toimintoon määritellään missä muodossa se palauttaa siitä saatavan käyttäjän syötteen. Ohjaussauvan kohdalla tarvittiin takaisin sen liikkuminen ylös, alas, oikealle ja vasemmalle. Näiden ominaisuuksien takia Vector2-tyyppinen muuttuja sopi tähän toteutukseen täydellisesti. Esimerkiksi pelatessa kun, ohjaussauvaa kääntää oikealle Vector2-tyyppinen muuttuja palauttaa arvona suunnan ja liikutusmäärän mihin suuntaan sitä on käännetty. Voimakkuus on tässä tapauksessa arvo nollan ja yhden väliltä. Kuvassa 19 Ohjaussauvan liikuttamisen palautusarvoksi valittu Vector2-tyyppinen muuttuja.



KUVA 19. Action Asset tiedosto.

6.3 Pelaajan liikuttaminen

Projektiin ennestään oli luotuna PlayerMovement luokka, jossa käyttäjän syöte otettiin vastaan. PlayerMovement luokan tarkoituksena on liikuttaa pelaajahahmoa pelissä. Muokattaessa koodia käyttäjäystävällisemmäksi päädyttiin erottamaan käyttäjän syötteen lukeminen ja pelaajan liikuttaminen eri luokkiin. Jatkossa PlayerMovement luokka vastaa pelkästään pelaajahahmon liikuttamisesta ja uusi PlayerInput luokka ottaa kaikki käyttäjän syötteet vastaan. Suurin syy tämän muutoksen tekemiseen oli lisättävä ohjaintuki puhelimen näytöllä oleville ohjaussauvoille.

PlayerInput luokassa etsittiin aktiivinen peliohjain, joka oli tässä tapauksessa virtuaalinen ohjaussauva ja kutsuttiin PlayerMovement luokan metodeja käyttäen ohjaussauvasta saatavia arvoja, jotka ohjasivat pelaajahahmoa.

6.4 Vaikeustason ja hauskuuden lisäys

Vaikeustasossa suurimpana ongelmana oli se, että pelaajan päästessä tasolle 10, alkoi vihollisia syntyä liian paljon kerrallaan. Syynä tähän oli se, että aikaisempien päätöksien mukaan haluttiin pelin pysyvän lyhyempänä. Ongelmaksi muodostui se, että peli oli aikaisemmin toteutettu tietokoneelle, jolla peliä oli helpompi pelata. Ratkaisuksi ongelmaan täytyi vaihtaa tasolla 10 syntyvien vihollisten määrää pienemmäksi.

Alkuperäisessä toteutuksessa pelaajan päästessä tasolle 10, peli aukaisi kaikki huoneet pelimaailmasta ja jokainen avattu huone lisäsi vihollismäärän kerrointa 0.25:llä. Järkevänä ratkaisuna päädyttiin alentamaan tämä kerroin 0.10:een, joka oli testien jälkeen toimivaksi todettu ratkaisu mobiilialustalle.

6.5 Pelaajan tähtäyksen toteuttaminen

Pelin työpöytä versiossa tähtääminen toimi hiiren ohjauksesta. Pelaaja tähtäsi siihen suuntaan missä hiiri oli pelaajasta nähden. Pelaajan koodiin piti tehdä muutos tähän, jotta pelaaja tähtäisi siihen suuntaan mihin ohjaimen oikeaa sau-

vaa käänsi. Tällä hetkellä ohjaussauvan liikkumisesta saatiin siis Vector2-tyyppinen arvo, joka osoittaa koordinaatistossa mihin suuntaan ohjaussauvaa käännetään. Ohjaussauvan palauttama Vector2-tyyppinen muuttuja palauttaa arvon (0,0), kun sitä ei liikuteta mihinkään suuntaan. Ohjaussauvan kääntäessä täysin vasemmalle Vector2-tyyppinen muuttuja palauttaa arvon (-1, 0). Se palauttaa myös arvot ääripäiden välistä, kun ohjaussauvaa ei käännetä loppuun asti.

Tähtäyssuunnan laskeminen ohjaussauvasta saadun arvon avulla. Ensiksi tallennetaan ohjaimesta saatu arvo Vector2-tyyppiseen lookDir-nimiseen muuttujaan. Tässä vaiheessa meillä on tiedossa koordinaatistossa mihin suuntaan ohjaussauvaa käännetään. Tavoitteena on saada kulman asteluku, johon pelaajahahmo käännetään. Tähän tarvitaan matemaattista kaavaa Atan2. Atan2 palauttaa x- ja y-koordinaattien arvojen argustangentin. Arvo saadaan radiaaneina ja se tulee muuttaa kulman asteluvuksi. Muutos tapahtuu Unity-pelimoottorilla kätevästi käyttäen Rad2Deg funktiota:

360 / (PI * 2)

Jos pelaajahahmo olisi toteutettu sillä tavalla, että se alussa katsoisi x-akselin mukaisesti niin lopussa ei olisi tarvinnut tehdä kulman kompensaatio vähennystä. Pelaajahahmo katsoi y-akselin mukaisesti alaspäin, joten tässä tapauksessa 90 asteen vähentäminen saadusta asteluvusta korjasi tämän ongelman. Kuvassa 20 näkyy matemaattisen kaavan Atan2 hyödyntäminen ja pelaajahahmon kulman kompensointi.

```
public void RotatePlayer(Vector2 rightStickValue)
{
    Vector2 lookDir = rightStickValue;

    float angle = Mathf.Atan2(lookDir.y, lookDir.x) * Mathf.Rad2Deg - angleOffset;
    playerRigidBody.rotation = angle;
}
```

KUVA 20. Tähtäyssuunnan laskeminen ohjaussauvasta saadusta Vector2-tyyppisen muuttujan arvosta.

6.6 Pelaajan ampuminen

Tietokoneversiossa ampuminen oli toteutettu hiiren vasemmasta painalluksesta. Käyttäjän painaessa hiiren vasenta painallusta, kutsuttiin skriptissä Gunsystem.cs metodia public void ShootingInputs(). Metodissa tarkistettiin, jos hiiren vasen näppäin oli pohjassa niin boolean-tyyppisen muuttujan shooting arvo muutettiin todeksi. Muuttujan shooting ollessa tosi metodia void Shoot() kutsuttiin, joka sai pelaajan ampumaan ammuksia.

Mobiiliversioon seuraavat muutokset tuli tehdä, jotta ampuminen saatiin toimimaan halutulla tavalla. Playerinput.cs skriptissä otetaan vastaan virtuaalisen ohjaussauvan käyttäjänsyöte. Skriptissä otetaan myös referenssi virtuaaliseen ohjaussauvaan, josta saadaan pelaajan tekemät liikkeet ohjaussauvaan talteen. Näillä tiedoilla kutsutaan PlayerMovement.cs skriptitiedoston metodia public void GetMovementInputs(Vector2 leftStickValue). Edellä mainittu metodi ottaa PlayerInput.cs skriptistä arvon Vector2-tyyppiseen muuttujaan leftStickvalue, joka antaa arvon kuinka paljon ohjaussauvaa on liikutettu. Vector2-tyyppisen muuttujan arvo sisältää kaksia arvoa, jotka voidaan ajatella koordinaatiston x- ja y-akselina. Ohjaussauvan liike vasemmalle alas tuottaa arvon (-1, -1), koordinaatistolla yksi alaspäin ja yksi vasemmalle. Arvot (0, 0) tarkoittaa, että ohjaussauvaa ei liikuteta mihinkään suuntaan. Arvot (1, 1) vastaavat sitä, että ohjaussauva on liikutettu oikealle ylös. Näiden arvojen avulla voidaan luoda helposti hiiren painallusta vastaava toiminto hyödyntäen sitä, kuinka pitkälle ohjaussauvaa on liikutettu tiettyyn suuntaan. Ohjaussauva palauttaa arvoja 0 ja 1 väliltä. Jos ohjaussauvaa on liikutettu puoleen väliin sen akselia, niin saadaan arvo 0.5. Tästä voidaan luoda puhelimelle hyvin sopiva tapa toteuttaa ampumispainallus sillä, kuinka paljon ohjaussauvaa on liikutettu. Haluamme saada pelaajan ampumaan, kun ohjaussauva on liikutettuna melkein täysin johonkin suuntaan. Jos laitamme ampumisherkkyyden arvoksi 0.8 niin viimeisellä 20 prosentin alueella ohjaussauvan liikkumisalueesta ampumismetodia kutsutaan ja pelaaja ampuu ammuksen.

Koska ohjaussauvasta saamamme arvot ovat Vector2 muodossa (0,0) – (1,1) meidän pitää laskea niiden yhteenlasketun yksittäisen vektorin pituus. Vektorin pituus saadaan kahden vektorin yhteenlasketusta summasta ja sen neliöjuuresta.

Tulokseksi laskutoimituksesta saadaan liukuluku 0 ja 1 väliltä kertoen miten paljon ohjaussauvaa on liikutettu. Kuvassa 21 on skriptistä lainattu kuva, josta näkee, että ohjaussauvan liikutuksen tietyssä vaiheessa sen arvo menee yli ennalta määritetystä arvosta, jonka jälkeen boolean-tyyppinen muuttuja shooting muuttuu todeksi ja metodia kutsutaan.

```
if (currentWeapon.allowButtonHold)
{
    shooting = shootValue.magnitude > shootSensitivity;
}
```

KUVA 21. Kooditoteutus, jonka avulla ampuminen tapahtuu, kun ohjaussauva liikkuu määritettyyn kohtaan.

6.7 Koodin rakenteen muutokset

Pelin työpöytä versiossa koodissa oli käytetty jonkin verran kovakoodattuja muuttujia. Näitä oli osittain jätetty toimintojen testaamista varten. Kuvassa 22 näkyy kovakoodattu arvo. Kovakoodattuja arvoja on vaikea käydä muuttamassa keskeltä pitkiä koodeja.

```
106            if (rightStickValue.magnitude > 0.2f)
```

KUVA 22. Kovakoodattu arvo 0.2f.

Kovakoodatut arvot pitää muuttaa muuttujiksi, jotka ovat helppo säätää ja ylläpitää tulevaisuudessa. Kuvassa 23 näkyy lisätty muuttuja korvaajaksi kovakoodatulle arvolle.

```
public float playerRotateSens;
```

KUVA 23. Uusi muuttuja kovakoodatulle arvolle.

Jos Unity-objektiin liitettyyn skriptiin luo julkisen muuttujan, se tulee näkyviin Unity editoriin. Näiden muuttujien arvo voidaan muokata Unity Editorista. Tämä helpottaa projektin ylläpidettävyyttä ja poistaa kovakoodatut muuttujat. Kuvasta 24 nähdään julkisen muuttujan säätövalikko Unity Editorista.

KUVA 24. Unity Editori näyttää objektiin liitetyn koodin julkiset muuttujat.

6.8 Muistinhallinta

Projektissa Object Poolia käytettiin pelaajan ammusten objektien säilyttämiseen. Object Poolia käyttäessä ammus objektit ladataan ennen pelin alkua valmiiksi tietokoneen muistiin ja niitä voidaan käyttää uudelleen sieltä. Unity-pelimoottorissa suurena kompastuskivenä suorituskykyyn on objektien nopea luonti ja poistaminen muistista. Object Poolin avulla tätä suorituskyky haittaa ei tule.

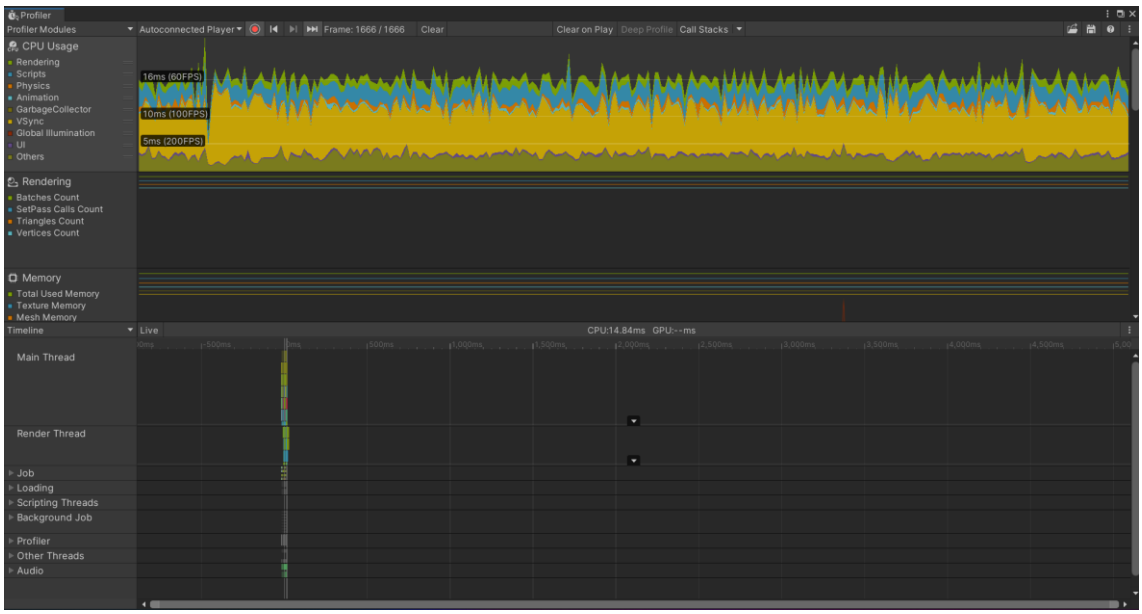
Projektissa luotiin ObjectPool.cs skripti, jonka tarkoituksena oli luoda kaikki object poolit ja sisältää toiminnallisuus objektien vapauttamiseen pooleista ja niiden takaisin laittaminen. Kun objekti vapautetaan, se aktivoidaan pelin sceneen ja sen takaisinlaitossa pooliin se de-aktivoidaan, jolloin se katoaa näkyvissä pelistä. Objekteja ei missään vaiheessa tuhota käyttäessä object poolia.

Ammuksiin myös tehtiin skripti ReturnToObjectPool.cs, jonka tarkoituksena oli pitää muistissa object poolin instanssi ja sen tagi, jolla se saatiin laitettua takaisin oikeaan object pooliin. Skriptissä toteutettiin myös objektin piilottaminen pelin scenestä.

6.9 Unity Profiler-analysointi

Projektissa käytettiin Unity-pelimoottoriin sisäänrakennettua suorituskyvyn seuraamiseen tarkoitettua työkalua nimeltään Unity Profiler. Työkalun avulla saatiin koontiversiota testatessa katsottua tarkemmin pelin eri osa-alueiden vaikutusta suorituskykyyn.

Profiler saatiin käyttöön laittamalla koontiversion asetuksista päälle Development Build ja Autoconnect Profiler. Näiden avulla Profiler saatiin aukeamaan samalla hetkellä tietokoneelle, kun koontiversio käynnistyi puhelimella. Profiler näyttää koontiversiosta ajantasaista dataa, jonka avulla saatiin paikannettua mahdollisia poikkeavuuksia suorituskyvyssä. Kuvassa 25 on kuvakaappaus Profiler-työkalun näkymästä koontiversiota testatessa.

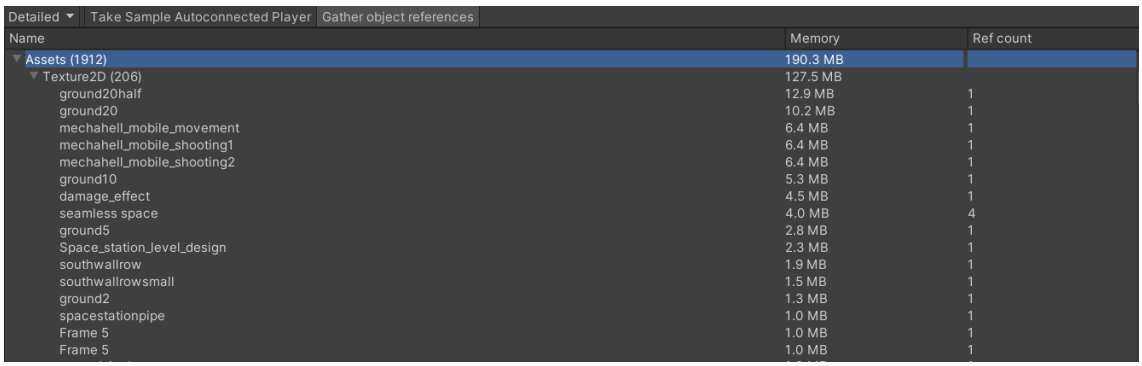


KUVA 25. Unity Profiler-työkalun kuvankaappaus koontiversion testaushetkellä.

Projektin aikana seurattiin tarkemmin prosessorin käyttöä sekä muistin käyttöä. Prosessorin käytön seurannassa on mahdollista nähdä hyvin yksityiskohtaisesti, että mitkä pelin osa-alueet varaavat eniten resursseja. Kuvassa 26 on kuvakaappaus Profilerista kun seurattiin miten paljon Unity-pelimoottorin Universal Render Pipeline varaa itselleen resursseja.

KUVA 26. Kuvakaappaus Unity Profilerista. Kuvasta näkee Universal Render Pipeline prosessorin käytön prosentteina.

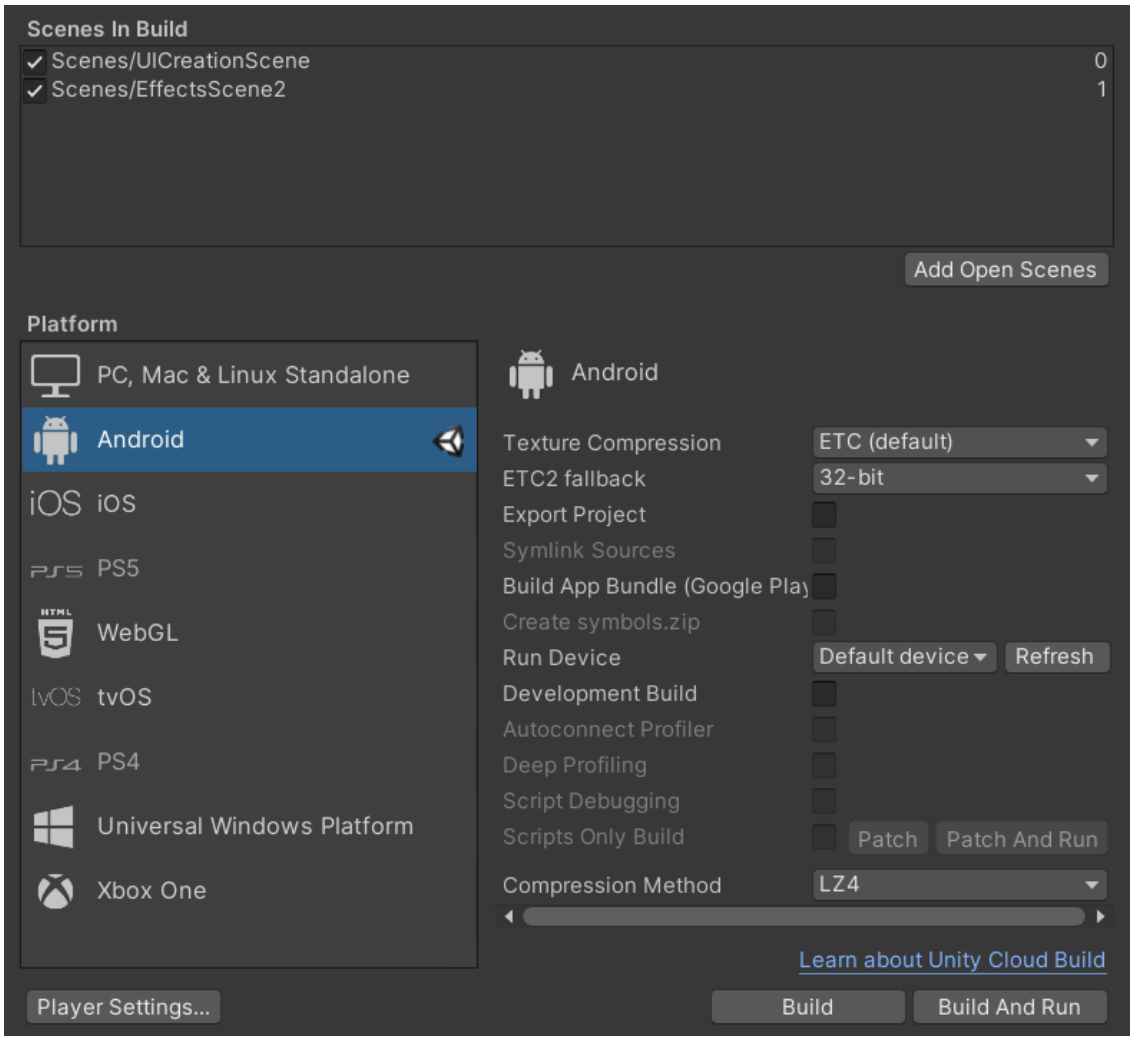
Muistinhallintaan Profiler tuli projektin aikana käytännölliseksi, sillä sen avulla saatiin havaittua yksityiskohtaisesti pelin tekstuurit, jotka vievät eniten muistia. Profilerin muistinkäytössä havaittiin tekstuureissa poikkeuksia ja Profilerilla kerätyn tiedon avulla poikkeukset saatiin korjattua. Kuvasta 27 näkee tekstuurit, jotka veivät eniten muistia.



KUVA 27. Unity Profiler-näkymä, josta näkee eniten muistia vievät tekstuurit.

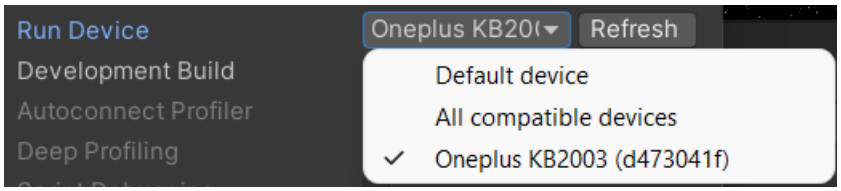
7 PROJEKTIN VIIMEISTELY

Projektin aikana pelistä tehtiin koontiversioita, joiden avulla pelin sai pelattavaan muotoon halutulla laitteella. Unityn koontiversion rakentaminen itsessään on hyvin minimalistinen prosessi. Koontiversioon valittiin pelin päävalikon sisältävä Scene ja itse pelin Scene. Koontiversion rakennuksessa on myös mahdollista vaikuttaa pelin kokoon asettamalla haluttu tekstuurien kompressointi. Tekstuurin kompressoinniksi valitsimme ETC-kompressoinnin, joka on Android koontiversioille oletusvaihtoehto. Kuvassa 28 on näkymä pelin koontiversion rakentamisen asetuksista. Jokaiselle alustalle on mahdollisuudet asettaa haluamansa asetukset. Myös halutut Scenet on mahdollista määrittää koontiversiolle.



KUVA 28. Kuvankaappaus pelin koontiversion rakentamisen ikkunasta.

Tarvittavat osat testaamiseen olivat itse puhelimet, USB-C tyypin kaapeli sekä tietokone, jolla pelin koontiversio rakennettiin. Puhelimen asetuksista täytyi olla asetus ”USB-virheenkorjaus” päällä, jotta pelin koontiversio saatiin USB-yhteyden avulla tietokoneelta suoraan puhelimeen. Unityn Build And Run ominaisuutta käyttäessä Unity rakensi projektista koontiversion suoraan puhelimelle USB-yhteyden avulla. Build And Run ominaisuutta käyttäessä täytyi koontiversion asetuksista varmistaa, että kentässä Run Device oli oikea laite valittuna. Kuvassa 29 on esimerkki siitä, kun OnePlus 8T oli koontiversion asetuksissa valittuna oikeaksi laitteeksi.



KUVA 29. Unity koontiversion asetuksissa oleva Run Device asetus. Asetuksen avulla voidaan valita tietokoneeseen kytketty laite, jolle pelin koontiversio halutaan avata suoraan.

Koontiversion testaamisen tarkoituksena oli varmistaa Unity editorissa tehtyjen muutoksien toimivuus, kun projekti muutettiin mobiilille pelattavaan muotoon. Kun editorissa toteutettiin muutoksia esimerkiksi pelaajahahmon liikuttamiseen, niin muutoksien jälkeen projekti vietiin koontiversiona mobiilille ja samat muutokset testattiin uudestaan. Lopullinen versio pelistä tulee olemaan myös koontiversio, jonka asennustiedosto julkaistaan itch.io nettisivulle ladattavaksi.

8 POHDINTA

Opinnäytetyön tavoitteena oli saada lisää kokemusta mobiilipelien valmistamisesta Unity-pelimoottoria hyödyntäen. Aiemmin valmistetun pelin siirtäminen mobiilille antoi tähän hyödyllisen mahdollisuuden, sillä uuden pelin valmistamiselle ei tarvinnut varata aikaa. Pelin siirtämisellä mobiilialustalle saatiin mahdollisuus keskittyä itse pelin pelattavuuden toteutukselle mobiililaitteella. Tavoitteena oli myös näyttää lukijalle huomioitavat osa-alueet sekä erilaiset tekniikat, joita voidaan käyttää pelin siirtämisessä mobiilille. Ottaen huomioon mobiilipelien markkinoiden jatkuvasti kasvavan koon oli tärkeää saada kyseisen alustan pelien valmistamisesta lisää kokemusta tulevaisuutta ajatellen.

Tuloksena oli pelin onnistunut siirto tietokoneelta mobiilille. Pelin mobiiliversion toteutukseen saatiin aiemman kokemuksen avulla kiinnitettyä tarkkaa huomiota asioihin, jotka olivat tärkeitä mobiililla sulavan pelattavuuden mahdollistamiseksi. Pelin tietokoneversion valmistuksessa huomattu potentiaali mobiiliversiolle saatiin työn aikana hyödynnettyä.

Työtä voidaan jatkossa käyttää esimerkkinä, kun toteutetaan muita mahdollisia siirtoprojekteja toiselta alustalta mobiilille. Mobiilipelien käytäntöjen opettelu työn aikana antaa myös hyvän osaamisen pohjan, kun toteutetaan täysin alusta alkaen mobiilille tuotettua peliä. Opinnäytetyöstä voi eniten saada hyötyä lukijat kenellä on tulossa vastaan mahdollinen mobiilipeliprojekti tai pelin siirtäminen mobiilille.

LÄHTEET

Dickinson, C. 2017. Unity 2017 Game Optimization - Second Edition. Birmingham: Packt Publishing, Limited. Luettu: 15.10.2022. https://learning.oreilly.com/library/view/unity-2017-game/9781788392365/

Finnegan, T. 2015. Learning unity Android game development: learn to create stunning Android games using Unity. Birmingham, England: Packt Publishing. Luettu 15.3.2022. https://search-ebscohost-com.libproxy.tuni.fi/login.aspx?direct=true&AuthType=cookie,ip,uid&db=e000xww&AN=986369&site=ehost-live&scope=site

Manning, J. 2017. Mobile game development with Unity: build once, deploy anywhere. Sebastopol, California : O'Reilly Media. Luettu 21.4.2022. https://learning.oreilly.com/library/view/mobile-game-development/9781491944738/?sso_link=yes&sso_link_from=tampere-university

Shekar, S. 2017. Mastering android game development with unity: exploit the advanced concepts of unity to develop intriguing, high-end android games. Birmingham, England: Packt Publishing. Luettu 15.3.2022. https://search-ebscohost-com.libproxy.tuni.fi/login.aspx?direct=true&AuthType=cookie,ip,uid&db=e000xww&AN=1526149&site=ehost-live&scope=site

Sinicki, A. 2017. Learn Unity for Android Game Development: A Guide to Game Design, Development, and Marketing. Berkeley, CA: Apress L. P. Luettu 15.3.2022. https://learning.oreilly.com/library/view/learn-unity-for/9781484227046/?sso_link=yes&sso_link_from=tampere-university

Takoordyal, K. 2020. Beginning Unity Android Game Development From Beginner to Pro. Berkeley, CA: Apress. Luettu 15.3.2022. https://learning.oreilly.com/library/view/beginning-unity-android/9781484260029/?sso_link=yes&sso_link_from=tampere-university

Unity. 2022. Creating games for Android. Luettu 8.10.2022. https://unity.com/solutions/mobile/android-game-development

Unity Documentation. 2022a. Optimizing Scripts. Luettu 21.3.2022. https://docs.unity3d.com/2020.1/Documentation/Manual/MobileOptimization-PracticalScriptingOptimizations.html

Unity Documentation. 2022b. Canvas. Luettu 12.4.2022. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html

Unity Documentation. 2022c. Input System https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/QuickStartGuide.html

Unity Documentation. 2022d. Event System. Luettu 12.4.2022. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/EventSystem.html

Unity Documentation. 2022e. Button. Luettu 13.4.2022. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-Button.html

Unity Documentation. 2022f. Animator controller. Luettu 13.4.2022. https://docs.unity3d.com/Manual/class-AnimatorController.html

Unity Documentation. 2022g. Render pipelines. Luettu 20.4.2022. https://docs.unity3d.com/Manual/render-pipelines.html

Unity Documentation. 2022h. Object Pool. Luettu 20.7.2022. https://docs.unity3d.com/ScriptReference/Pool.ObjectPool_1.html

Unity Documentation. 2022i. Universal Render Pipeline. Luettu 20.07.2022 https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.10/manual/index.html