OULUN AMMATTIKORKEAKOULU

Mikko Korpela

LAPLAND RUNNER -PELIN KEHITYS UNITY-PELIMOOTTORILLA

LAPLAND RUNNER -PELIN KEHITYS UNITY-PELIMOOTTORILLA

Mikko Korpela
Opinnäytetyö
Kevät 2019
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä: Mikko Korpela
Opinnäytetyön nimi: Lapland Runner -pelin kehitys Unity-pelimoottorilla
Työn ohjaaja: Kari Laitinen, Niko Alakastari
Työn valmistumislukukausi ja -vuosi: Kevät 2019 Sivumäärä: 31 + 4 liitettä

Opinnäytetyö tehtiin *Lapland Runner* -pelin kehityksen yhteydellä Frozen Visionille. Työn tavoitteena oli saada toimiva peli asiakkaan vaatimusten mukaan. Työ käsittelee Unity-editoria, pelikonseptia, pelin suunnittelua, ohjelmointia, toteuttamista, optimointia ja testausta.

Työssä käytettiin Unity-pelin kehitysympäristöä, jonka avulla rakennettiin 1–4 pelaajalla pelattava loputon juoksijapeli. Pelaajilla on pelattavana loputtomasti generoituva pelikenttä. Pelikentän loputon generointi on kehitetty pelikentän paloja kierrättämällä, jotta pelistä saadaan mahdollisimman suorituskykyinen. Pelissä erilaista on se, että normaalien ohjaimien sijaan pelissä käytettiin kontrollerina tasapainolautaa, jonka avulla pelihahmoa liikutellaan.

Lopputuloksena oli tilaajalle sopiva peli, joka täytti tarpeelliset vaatimukset. Työssä tehtiin hyvä pohja pelille, jota voidaan jatkokehittää tarpeen tullen.

Asiasanat: Unity, pelin kehitys, ohjelmointi, C#, Object Pooling

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Degree Programme of Software Development

---

Author: Mikko Korpela
Title of thesis: Lapland Runner game development with Unity engine
Supervisor(s): Kari Laitinen, Niko Alakastari
Term and year when the thesis was submitted: Spring 2019 Number of pages: 31 + 4 appendices

---

This Thesis was made along with the game development of *Lapland Runner* game for Frozen Vision. The goal of this thesis was to make a working game for the needs of the customer. This thesis adresses game development for these parts: Unity Editor, game consept, game design, programming, assembling, optimization and testing

Unity game development software was used to create 1–4 multiplayer game with infinite runner mechanics. The players are able to play on an endlessly generating level. Endless level design was developed using a technique that recycles the map pieces. Recycling game these map pieces allows for a high performance. What was unique about the game that, instead of the usual controller, players needed to use a balance board to move their characters around the map.

The result of this thesis was a suitable game which completed the needs of the customer, and it has all the requirements for further development if needed.

Keywords: Unity, Game Development, Programming, C#, Object Pooling

SISÄLLYS

TERMILUETTELO

2d	Two-dimensional, kaksiulotteinen
3d	Three-dimensional, kolmiulotteinen
Arduino	Elektroniikka-alusta ja ohjelmointiympäristö, johon voi kytkeä erilaisia komponentteja kuten gyroskoopin
Asset	Pelin käyttämät ulkoiset resurssit esim. grafiikka, äänet tai skripti
Backlog	Tuotteen kehitysjono
Bitbucket	Versionhallinnan säilytyksen pilvipalvelu
Branch	Versionhallinnan haarake
Collider	Peliobjektin fyysinen osa, joka tutkii törmäyksiä
Float	Muuttuja, joka pitää arvot desimaalien tarkkuudella
FPS	Frames per second, kuvataajuus, montako kuvaa sekunnissa peli toistaa
Frame	Yksi kuva pelissä
Git	Versionhallinta järjestelmätyökalu
Gyro	Gyroskooppi, joka tarkkailee liikkeen säilymistä sitä kallistaessa ja kiertäessä
Infinite runner	Peli, jossa pelihahmo juoksee eteenpäin loputonta pelikenttää
Monodevelop	Unityn vaihtoehtoinen ohjelmointi-kehitysympäristö
MVP	Minimum Viable Product. Ensimmäinen asiakkaalle jaettava versio
Object Pooling	Peliobjektien kierrätystapa tuhoamatta ja synnyttämättä niitä uudelleen
Prefab	Valmis konfiguroitu peliobjekti
Rigidbody	Pelikomponentin fyysinen simulaatio. Toimii Unityn pelimoottorilla
Scene	Sisältää yhden pelinäkymän peliobjektit
Script	Skripti, joka suorittaa jotain tiettyä tehtävää
Shader	Skripti, joka laskee materiaalien värejä tai taittoja
Task	Projektin aikana suoritettava yksittäinen tehtävä tai työ
Unity	Unity Technologiesin kehittämä pelimoottori
Visual Studio	Microsoftin ohjelmointikehitysympäristö

1 JOHDANTO

Opinnäytetyössä oli tarkoitus kehittää toimiva peli Frozen Vision -yrityksen asiakkaalle käyttäen tasapainolautaa peliohjaimena. Peli on suunniteltu täysin käyttäen Unity-pelimoottoria. Peliohjaimena toiminut tasapainolauta oli liitetty Windows-pohjaiseen pöytätietokoneeseen COM-portin avulla.

Opinnäytetyö kirjoitettiin osittain pelin kehityksen aikana, mutta viimeisteltiin projektin valmistuttua. Työ käsittelee Lapland Runner -pelin kehityksen vaiheita ja sen toteuttamista. Peli on ns. infinite runner -tyyppinen peli, jossa pelaaja liikkuu loputtomalla pelikentällä. Työssä kehitettiin valmis peli tilaajalle, joka voi tarpeen mukaan myös jatkokehittää sitä.

Työ käsittelee pelin kehitystä Unity-pelimoottorilla, pelin suunnittelua ja sen dokumentointia. Pelin Suunnittelu-luvussa käsitellään pelin vaatimuksia ja tarpeita sekä mahdollisuuksia. Tässä vaiheessa käydään myös läpi tarvittavat työkalut, kuten versionhallinnan ja dokumentoinnin työkalut. Pelin mekaniikkoja, tekniikkaa ja ohjelmointia käsittelee Ohjelmointi-luku. Työn graafinen puoli sekä animaatiot käsitellään Toteutus-luvussa.

Lopussa käsitellään työn onnistumista ja soveltuvuutta asiakkaan tarkoitukseen. Opinnäytetyön tavoitteena on dokumentoida pelin suunnittelu- ja kehitysprosessit.

2 UNITY -EDITORI

Unity on laaja ja monipuolinen pelin kehitysalusta kaikille kehittäjille. Unity on Unity Technologiesin kehittämä pelimoottori, joka julkaistiin vuonna 2005. Unity sisältää mahdollisuuden kehittää interaktiivisia pelejä 2d- ja 3d-ulottuvuuksilla. Unityn editori on suoraviivainen, jonka takia se on erinomainen aloittelijoille ja kokeneille kehittäjille. Unity on rakennettu helppokäyttöisen editorin pohjalle, mutta se kuitenkin sisältää laajan tuen skripteille, jotka parantavat ohjelmointimahdollisuuksia. Unityn ohjelmointieditorina toimii Microsoft Visual Studio, mutta ennen tätä Unity käytti MonoDevelop-ohjelmaa (Paczkowski 2018).

Unity aloitti vuonna 2005 pelkästään Mac-tuella, mutta tästä alle vuoden sisään lisättiin Windows-tuki. Unity on myös vuosien saatossa lisännyt monia julkaisualustoja. Näitä julkaisualustoja ovat mm. Web, IPhone, Android, Ps4 ja Xbox.

Unity on saatavilla yksityisille ilmaiseksi, mutta mahdollistaa myös päivityksen Unity Plus- ja Unity Pro -paketteihin. Yrityksille Unity ei ole ilmainen, vaan sen on maksettava Unity Pro -paketin hinta.

3 PELIKONSEPTI

Pelini sai nimensä Lapland Runner vasta pelin kehityksen loppuvaiheilla. Tätä ennen peliä kutsuttiin yksinkertaisesti nimellä Project Lapland. Lapland Runner on peli, jonka pelialusta on PC. Peliohjaimena yleisen näppäimistön ja hiiren sijaan on tasapainolauta, jonka alla on tietokoneeseen liitetty Arduino-pohjainen gyroanturilla toimiva järjestelmä. Ohjaimella on tarkoitus liikkua seisomalla tasapainolaudan päällä kallistaen lautaa ja väistellä pelissä tulevia erilaisia esteitä päästäkseen mahdollisimman pitkälle.

Pelin vaikutteita ovat pääasiassa infinite runner -pelit (esim. *Subway Surfer*). Graafiset vaikutteet näkyvät pelikonseptissa (kuva 1).

KUVA 1. Taiteelliset vaikutteet pelikonseptissa

Pelissä on pelimekaniikkoja, joiden avulla täytyy selvitä mahdollisimman pitkälle. Tasapainolaudalla ohjataan sivuttaisliikkeitä, jottei törmätä radalla liikkuviin lukuisiin esteisiin. Esteet ovat luotu satunnaisesti noin 30 eri yhdistelmästä. Täysin satunnainen radan luominen voisi johtaa läpipääsemättömään tilanteeseen. Yhdistelmäesteet poistavat mahdottomien tilanteiden muodostumisen ja avaavat mahdollisuuden pelata niin pitkälle kuin pystyy. Pelissä häviäminen tapahtuu törmäämällä esteisiin, joihin menee yksi elämä. Pelaajat aloittavat pelin viidellä elämällä, ja kun elämät kuluvat pois, on peli ohi.

Peliä on mahdollista pelata 1–4 pelaajan porukoissa, jolloin saavutetaan kilpailuhenkeä ja tavoitellaan hyvän ilmapiirin kehitystä. Jokainen pelaaja voi valita oman pelihahmon, jolla he pelaavat seuraavan pelin (kuva 2). Pelin lopussa näkyvä pelin pisteytys lasketaan peliajan avulla ennen eliminointia.

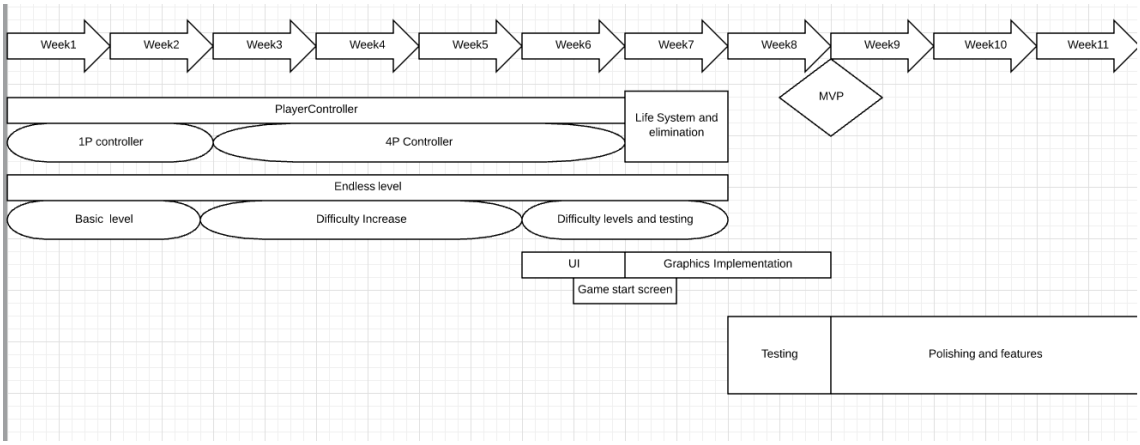
KUVA 2. Peli nelinpelinä

Tasapainolaudalla pelihahmon ja pelivalikkojen kontrolloiminen oli kriittinen osa pelin kehitystä, koska pelaajat eivät pääse käsiksi näppäimistöön ja hiireen.

Pelin toiminta perustuu Infinite runner -pelityyliin, että pelihahmot juoksisivat koko ajan eteenpäin kartassa. Pelikenttä on koottu eri liikkuvista ja kierrätetyistä yhdistelmäpalasista. Yhdistelmäpalaset generoivat pelikenttää loputtomasti. Tätä ohjelmointitapaa kutsutaan nimellä Object Pooling.

4 PELIN SUUNNITTELU

Projektissa oli tarkoitus luoda helposti pelattava peli tietyillä asetuksilla, jotta siihen olisi helppo päästä mukaan. Peli on suunniteltu pelattavaksi kaikenikäisille, joten se ei saa olla liian monimutkainen. Projektin alussa suunniteltiin eri osa-alueille tiukat aikamääreet, joihin tulisi pyrkiä. Projektiin suunniteltiin myös MVP eli pienin julkaisukelpoinen tuote (kuva 3), jonka avulla hahmotettiin, kuinka paljon aikaa eri kokonaisuuksiin tulisi käyttää. Kyseessä oli ehdoton minimi, johon projektin aikana tuli pyrkiä. Tämän avulla varmistettiin, että projektista tulee kokonaisuutena valmis eikä käytetty liikaa aikaa pieniin osuuksiin.



KUVA 3. MVP eli Minimum Viable Product -suunnitelma

Projektin suunnittelussa on tärkeää käydä läpi kaikki mahdolliset vaiheet ja tehdä niille aika-arviot. Tärkeää on myös suunnitella, että mitkä osat kehitetään omalla tiimillä ja tarvitaanko ulkoisia assetteja, eli valmiita pelin käyttämiä ulkoisia resursseja, kuten grafiikka, äänet tai skripti.

Projektin kivijalka on projektin alussa laadittu pelinsuunnitteludokumentti. Kaikki peliin liittyvät tärkeät asiat tulisi listata pelisuunnitteludokumenttiin samaan aikaan tai ennen kuin ne toteutetaan peliprojektin kehityksessä. Dokumentti ei kuitenkaan ole täydellinen ohje pelin kehitykseen, vaan sen pitäisi sisältää pääpiirteet pelistä. Pelin pääpiirteet ovat yleisesti sen kokonaisvisio, taiteellinen suunta ja peliominaisuudet (Gamedesigning 2018).

4.1 Käyttöliittymä

Pelin käyttöliittymän suunnittelu on haasteellinen monella tapaa. Suurimmat haasteet tämän projektin käyttöliittymän kehittämiseen liittyivät tasapainolauta-ohjaimen rajallisuuteen. Laudalla tulisi ohjata koko käyttöliittymää ja muita pelinsisäisiä valikoita ja painikkeita, kuten pelin pysäytys, jatkaminen ja uudelleenkäynnistäminen. Laudalla on periaatteessa neljä eri nappia. Nämä aktivoidaan kallistamalla lautaa eri suuntiin ja pitämällä sitä kaltevassa asennossa yhden sekunnin ajan. Tärkeää on myös määritellä Unityn pelinäkymät eli scenet, jotka sisältävät eri osa-alueet pelistä, kuten päävalikon, pelinäkymän, aloitusnäkymän ja voittoruudun (kuva 4). Scenejen hallinta tapahtuu Unityn pelimoottorin sisään rakennetun Scene Manager -luokan kautta.

KUVA 4. Scenet pelissä

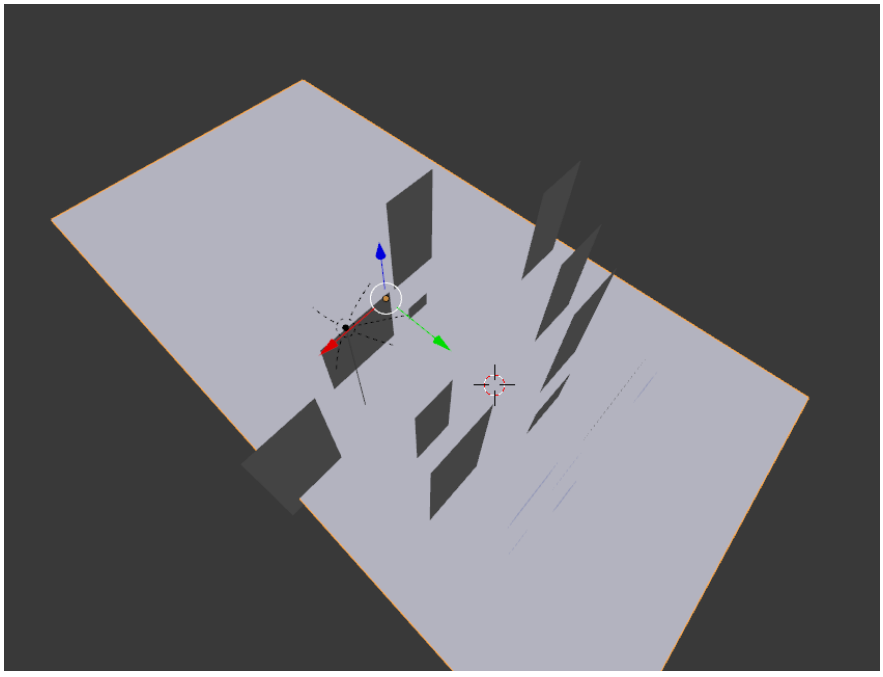
4.2 Gyroskooppi-ohjain

Gyroskooppi-ohjain projektiin suunniteltiin ilman tasapainolautaa. Arduinossa kiinni oleva gyroanturi tuli sovittaa Unityyn COM-porttien kautta, jonka avulla ohjelmoitiin tarvittavat muuttuja arvojen invertoinnit ja kalibroinnit. Arvojen Invertoinnit olivat tarpeellisia koska gyro-anturi oli ohjaimen alla ylösalaisin. Periaatteessa peli ei ikinä tarvitse tasapainolautaa, vaan sen pohjaan on kiinnitetty Arduino ja gyroanturi, joka hoitaa kaikki mittaukset tasapainolaudan liikkeiden mukaan. Tasapainolauta on ihan normaali lauta ilman mitään elektronista komponenttia, lukuun ottamatta laudan pohjaan kiinnitettyä Arduinoa. Arduinoa ohjaava SerialPortInterface-skripti, jonka päälle kehitettiin uusi anturin arvoja ohjaava PlayFloorController-skripti. PlayFloorController-skripti lukee arvot suoraan

SerialportInterface-skriptistä ja puristaa arvot maksimissaan ennalta määritettyihin arvoihin tasaisen kiihtyvyyden vuoksi.

4.3 Kenttäsuunnittelu

Suurin haaste kenttäsuunnittelussa oli rakentaa loputon kenttä lähestulkoon rajattomilla esteyhdistelmillä, mutta kuitenkaan luomatta ylitsepääsemätöntä esteyhdistelmää. Täysin satunnainen esteiden luominen oli siis poissa pelistä heti alussa. Esteiden luominen korjattiin tekemällä hieman pitempiä esteyhdistelmiä, joiden alku- ja loppupisteet eivät voi olla päällekkäin siten, että esteyhdistelmä olisi mahdoton. Kenttäsuunnittelun aloitus yksinkertaisimmillaan on asettaa peliobjekteja kentälle pelinäkymän alkuvaiheessa (kuva 5). Kenttäsuunnittelun konsepti edesauttaa myös sen graafisen puolen ideointia, koska graafikon tulee tietää, millaiseen kokonaisuuteen grafiikka tulee piirtää.



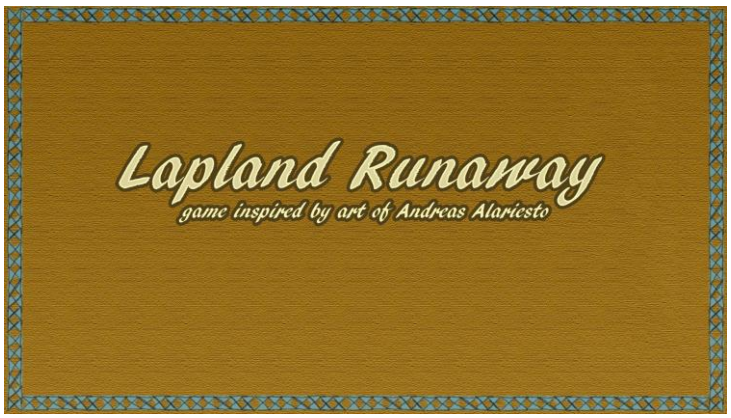
KUVA 5. Ensisuunnitelma pelikentän esteistä

4.4 Graafinen suunnittelu

Projektin aloitusvaiheessa oli tiedossa, että graafinen toteutus saa vaikutteita Andreas Alarieston Lappi-aiheisesta taiteesta. Graafinen suunta oli siis selkeä, joten peliobjektit saivat esteiksi mm. poroja (kuva 6), kiviä, kuusia, tukkeja ja pieniä majoja.

KUVA 6. Peliin suunniteltu Poroeste

Graafista suuntaa sai myös pelivalikot ja aloitusruutu (kuva 7).



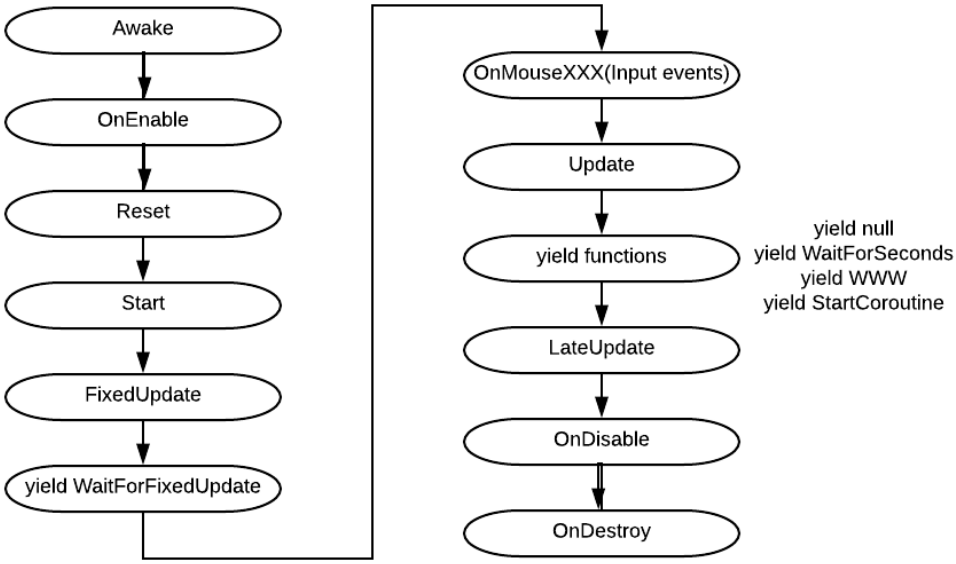
KUVA 7. Pelin aloitusruutu

4.5 Versionhallinta

Git toimi versionhallinan ensisijaisena työkaluna. Gitin avulla projektin tallennus Gitlab-säilöön toimi vaivattomasti ja sen avulla projekti jaettiin useampaan tallennus haaraan eli branchiin. Versionhallinta ohjelmistoprojektissa on erittäin tärkeää, koska kehityksessä voi tapahtua virheitä, joiden vuoksi paluu aikaisempaan versioon on suotavaa. Versionhallinnan työkalut mahdollistavat usean kehittäjän yhteistyön samaan projektiin. Itse ohjelmistoprojektin versionhallinnan lisäksi käytin myös Trello- ja Atlassian Jira -ohjelmistoja yksittäisten tehtävien hallintaan ja kehitysjonona eli backlogina.

5 OHJELMOINTI

Peliohjelmointi Unityssä koostuu eri skripteistä, jotka ajavat pelin eri komponentteja. Ohjelmointikielenä Unityn skripteissä toimii C#. Ohjelmoitaessa Unityllä on tärkeä tietää, miten sisäänrakennettujen luokkien kuten Start, Update ja FixedUpdate päivitysjärjestys toimii.
Skriptin elämänkaaren pääfunktioiden yleiskatsaus sisältää tärkeimmät skriptin funktiot (Unity Technologies 2019a). Funktioiden välissä on myös useita kuvassa näkymättömiä funktioita (kuva 8), jotka ajavat eri asioita, kuten pelin fysiikkaa, logiikkaa ja kuvan renderöintiä.



KUVA 8. Skriptin funktioiden yleiskatsaus Unityllä skriptin herättämisestä tuhoamiseen.

Päivitysfunktioita on kolmea erilaista ja niille kaikille on oma käyttötarkoitus. Suurimmat eroavaisuudet päivitysfunktioissa ovat seuraavat.

FixedUpdate-funktiota kutsutaan useammin kuin Update-funktiota. Sitä voidaan kutsua useasti yhden kuvan aikana. Tätä päivitysfunktiota käytetään yleensä pelaajan fyysiseen liikuttamiseen.

Update-funktiota kutsutaan yhden kerran kuvan aikana. Tämä funktio yleensä sisältää suurimman osan pelinaikana tehtävistä ja laskuista.

LateUpdate-funktiota kutsutaan, kun Update-funktio on täysin valmis. Funktio on erittäin käytännöllinen esimeriksi pelaajaa seuraavalle kameralle.

Pelissä on myös objekti nimeltä GameManagerAll, joka ei tuhoudu scenejen vaihduttua. Tämän objektin avulla voi siirtää arvoja kuten pelaajamäärä ja tulokset scenestä toiseen. Uuden scenen lataaminen tuhoaa kaikki pelikomponentit, mutta sen voi kiertää kutsumalla Unityn DontDestroyOnLoad sisäänrakennettua funktiota (Unity Technologies 2019b).

5.1 Pelaajakontrollit

Pelihahmon kontrollit ovat hyvin yksinkertaiset. Pelaaja voi liikkua vain vasemmalle tai oikealle (kuva 9). Pelaajakontrolli-skriptissä on myös muita ominaisuuksia, kuten liikkumisrajoitukset ja pelaajan collideri, joka tarkastelee törmäyksiä kartan muihin peliobjekteihin, jotka ovat luokiteltu tappaviksi. Skripti sisältää myös uudelleenluonti-funktion, joka ajetaan pelaajan törmätessä tappavaan objektiin. Pelaajan uudelleenluonti-funktio antaa pelaajalle 5 sekunnin kuolemattomuuden ja nollaa pelihahmon sijainnin pelikentän keskelle.

```
private void Update()
{

    if (_playFloorController.clampedRoll > rollMinThreshold)
    {
        invertedRoll = Mathf.Abs(_playFloorController.clampedRoll);
        Debug.Log(invertedRoll);
    }


    else if (_playFloorController.clampedRoll < -rollMinThreshold)
    {

        invertedRoll = -Mathf.Abs(_playFloorController.clampedRoll);
    }

    else
    {
        invertedRoll = 0;
    }

    if (controller.isGrounded)
    {
        //Gyro Control
        if (gameManagerScript.isGyroActive)
            moveDirection = new Vector3(invertedRoll, 0, verticalPlayerSpeed);
        else
            moveDirection = new Vector3(Input.GetAxis("HorizontalP" + playerEnd), 0, verticalPlayerSpeed);
        moveDirection = transform.TransformDirection(moveDirection);
        moveDirection.x *= horizontalPlayerSpeed;
        moveDirection.z *= verticalPlayerSpeed;

        if (_playFloorController.clampedPitch > 0.8f)
            moveDirection.y = jumpSpeed;
    }
    moveDirection.y -= gravity * Time.deltaTime;
    controller.Move(moveDirection * Time.deltaTime);

    //move player back if player crosses a border
    if (player.transform.position.x > border)
        player.transform.position = new Vector3(border, player.transform.position.y, player.transform.position.z);
    if (player.transform.position.x < -border)
        player.transform.position = new Vector3(-border, player.transform.position.y, player.transform.position.z);
}
```

KUVA 9. PlayerController.cs-skriptin Update-funktio

Kun scene on latautunut pelin alussa, pelinhallintaobjektin Pelinhallinta-skripti (GameManagerScript) saa tarvittavat pelaajatiedot GameManagerAll -pelikomponentin skripteiltä. Pelinhallinta-skriptien avulla toteutetaan oikea määrä pelihahmoja pelaajille (Unity Technologies 2019c). Pelikamera kulkee aina ensimmäisen pelaajan perässä (player1), ellei pelaaja1 eliminoidu ennen muita pelaajia, jolloin se hyppää seuraavaan hengissä olevaan pelaajaan. Seurattavan pelaajan vaihtuminen ei aiheuta pelissä kameran hyppäämistä, koska kaikki pelaajat juoksevat samalla z-akselilla ja kamera on ohjelmoitu seuraamaan vain sillä akselilla, muiden akselien ollessa lukossa.

5.2 Moninpeli

Moninpeliin toteutettujen pelaajien määrä riippuu siitä, että montako pelihahmoa pelin aloitusvalikossa valitaan. Peli lataa kaikille pelaajille Resources/Players-kansiosta valmiin Player prefabin, joka on valmiiksi konfiguroitu peliobjekti pelaajalle. Player-prefab kopioidaan sille pelaajalle ja eritellään muista pelaajista nimeämällä se Player1–4.

Kaikkien pelaajien toteutus ja nimeäminen tapahtuu yksinkertaisella foreach-silmukalla (kuva 10). Luuppi avulla lisätään pelihahmoille myös kaikki tarvittavat komponentit ja skriptit.

```
private void PlayersInstantiate()
{
    foreach (var p in activePlayers)
    {
        activePlayers[playerNumber] = p;
        SydanSpawn(playerNumber);

        playerNumber++;

        newPlayer = Instantiate(Resources.Load("Players/Player" + playerNumber), new Vector3(xPos, 2f, -2),
            Quaternion.identity) as GameObject;
        newPlayer.name = "Player" + playerNumber;
        xPos++;
    }


    mainCamera = GameObject.FindGameObjectWithTag("MainCamera");
    mainCamera.AddComponent<CameraController>();
}
```

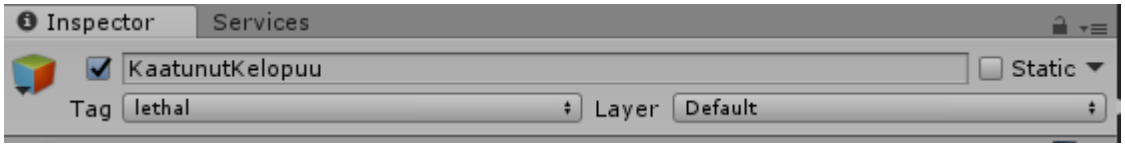
KUVA 10. Pelaajien toteutus

5.3 Pelaajien eliminointi

Törmätessään kentällä olevaan esteeseen pelaaja menettää yhden elämän. Menettäessä kaikki viisi elämää pelaaja eliminoidaan. Kaikkien pelaajien eliminoituessa peli päättyy. Pelaajien elämät näkee pelin laidassa sydäminä (kuva 11).

KUVA 11. Pelaajien elämät

Pelikentän esteobjekteille on annettu peliobjektia kuvastava collider, joka on merkattu kuolettavaksi pelaajille. Tagien avulla tiettyjen tagiparien törmäyksillä saadaan kutsuttua eliminointi funktioita (kuva 12).



KUVA 12. Tagit esteissä

Jokaisella pelaajalla on pelaajan toteutuksen yhdessä pelaaja-prefabin mukana tullut elämäskripti, joka laskee pelaajan elämät ja niiden menetykset pelaajan eliminointiin asti.

5.4 Kenttäkontrollit

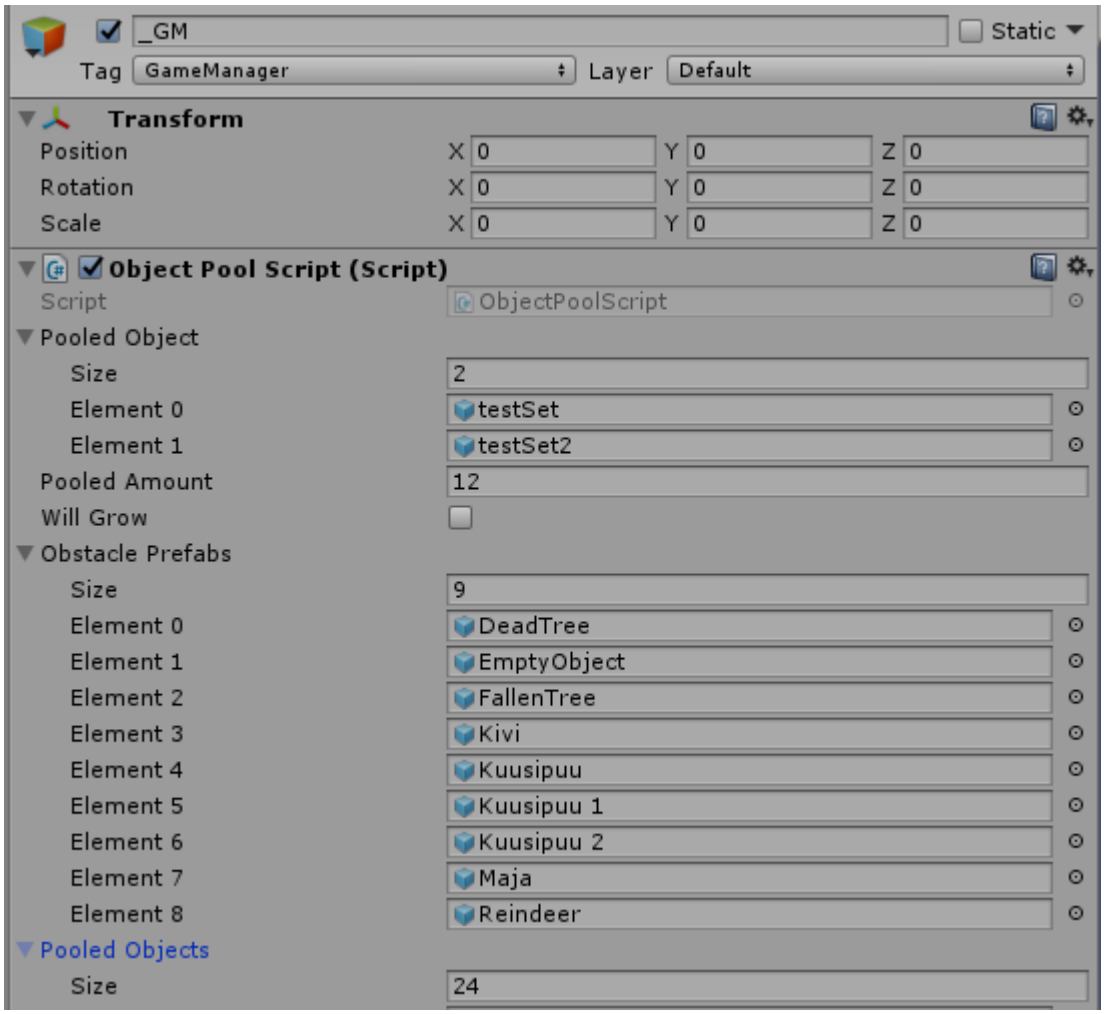
Pelin nimen mukaisesti kentän pituus on loputon, joten valmiiksi rakennetut kentät ovat mahdottomia tehdä. Pelikentän yhdistelmäesteet generoidaan pelin alussa, jonka jälkeen ne laitetaan satunnaisjärjestykseen toistensa perään. Kenttä generoituu sitä mukaa, kun pelaaja juoksee eteenpäin, mutta myös kentän alussa on 20 staattista palasta, jonka avulla saadaan pelaajan ja kentän generoinnin välille puskuri. Staattiset alkupalat tuhotaan 15 sekunnin päästä pelin aloituksesta, jolloin näitä palasia ei enää tarvita. Jokaisella generoidulla esteyhdistelmällä on esteyhdistelmien uudelleenjärjestys -skripti (liite 1). Liikuttelemalla esteitä saadaan pelikentälle lisää satunnaisuutta ja vähennämme pelikentän ennakoitavuutta.

5.4.1 Object Pool

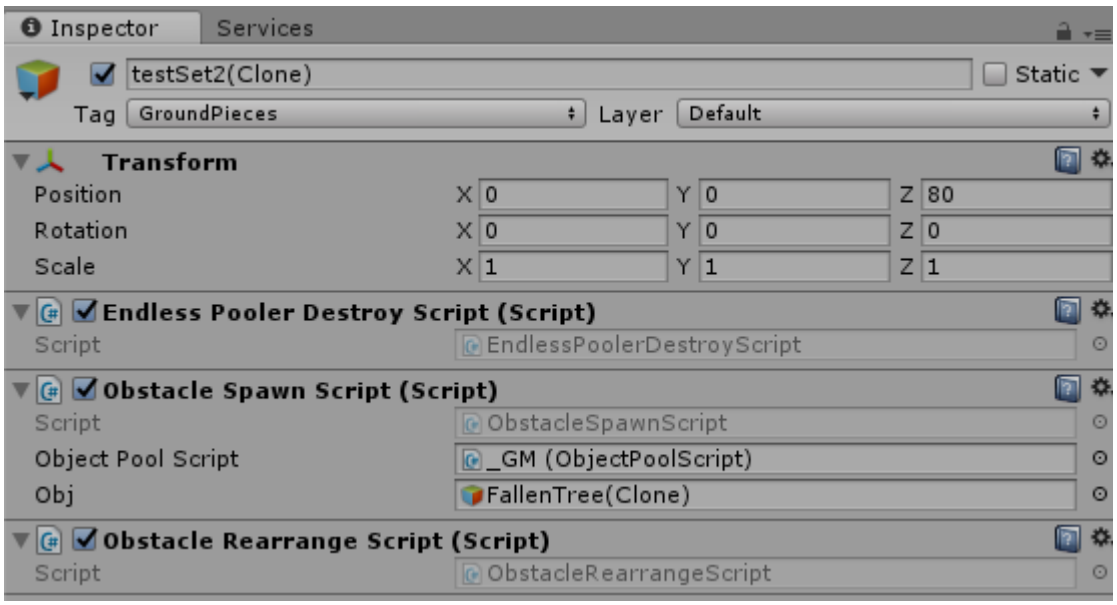
Object Pool tarkoittaa yksinkertaisimmillaan objektien kierrättämistä. Peliobjektien kierrätys on paljon kevyempi prosessi kuin niiden luominen ja tuhoaminen. Yleisesti Object Pool -tekniikkaa käytetään aina kaikissa pelin osa-alueissa, joissa tarvitaan jatkuvasti uusia peliobjekteja (Geig 2014). Object Pool kierrättää peliobjekteja luomalla alussa tarpeeksi peliobjekteja, joita aktivoidaan ja deaktivoidaan tarpeen mukaan. Deaktivoinnin yhteydessä peliobjektia ei siis tuhota, vaan se muuttuu näkymättömäksi ja toimettomaksi sen ajaksi, kunnes sitä taas kutsutaan uudestaan.

Projektissani käytän Object Pool -tekniikkaa kahdessa eri paikkaa. Suurin osa objektien kierrätyksestä tapahtuu pelikentän generoinnissa, mutta myös liikkuva horisontti käyttää samaa tekniikkaa, jonka avulla pelin taustaan saadaan liikkuvuuden tuntua.

Kaikki Object Pool -tekniikkaan kokonaisvaltaiset skriptit on liitetty näkymättömään pelinhallinta objektiin nimeltä *_GM*. Tämän objektin avulla ladataan resursseista esteyhdistelmä-prefabit ja sen avulla määritetään Object Poolin koko (kuva 13). Jokainen yksilöllinen toteutettu kenttäpalaklooni sisältää palan deaktivointiskriptit ja esteiden toteutukset ja niiden liikuttamiset (kuva 14).

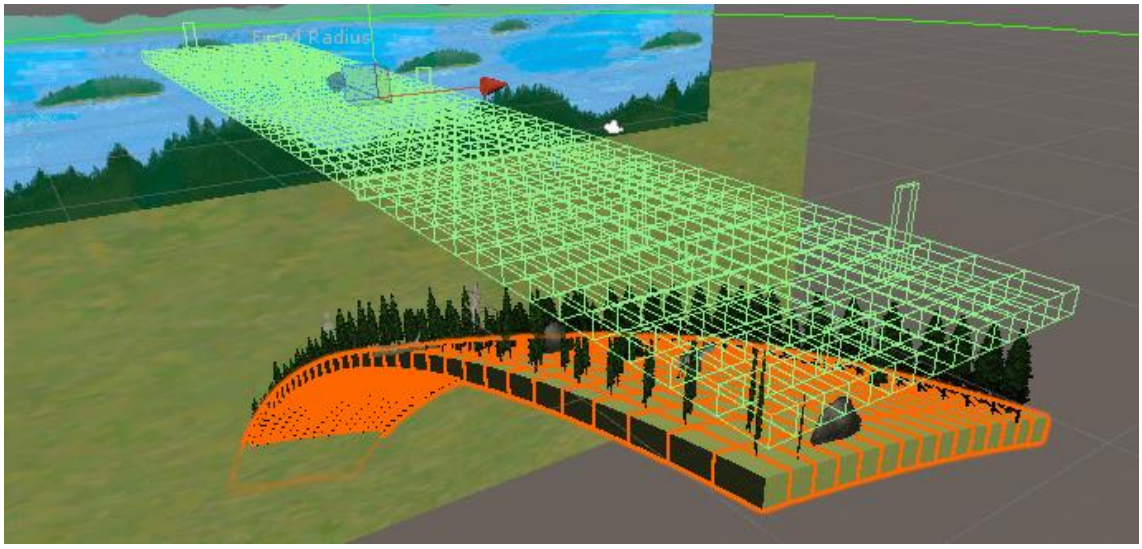


KUVA 13. Pelinhallinta prefabin Object Pool -skripti



KUVA 14. Kenttäpalanen ja siihen sisältyvät skriptit.

Object Poolin rakennukseen tarpeellinen skripti on ObjectPoolScript (liite 2), mutta sen käyttöä tämä kyseinen skripti ei hoida. Object poolin käyttöön tarvitaan uusi skripti (liite 3), joka aktivoi ja deaktivoi palasia poolista. Poolin palaset aktivoidaan sitä mukaan, kun pelihahmo juoksee eteenpäin (liite 4). Pelipalasten toteutus tapahtuu horisontin kaarron takana, jotta sitä ei näe pelin aikana. Peliobjektien fyysinen sijainti on suora, mutta horisontin taiton avulla pelipalaset generoidaan kameran ulkopuolella (kuva 15).



KUVA 15. Horisontin taitto avustamassa pelikentän generoinnissa

5.4.2 Liikkuvat esteet

Lapin poro on pelikentän laidalta laidalle liikkuva este. Poron mahdollisuus ilmestyä esteeksi on sama kuin muillakin esteillä eli 1/9. Porolle on lisätty rigidbody-komponentti, jonka avulla lasketaan yksittäisen peliobjektin fysiikkaa. Rigidbodyn kaikki muut liikkumaradat ovat lukossa paitsi liikkuminen x-akselilla. Poroeste saa toteutuksen jälkeen työntöä x-akselilla, jonka avulla se liikkuu toiseen laitaan. Poron toteutukseen ei pelkästään riittänyt sen luonti ja liikutus, vaan pelikentän sivussa olevat aidan prefabit tuli rikkoa, jotta porolla olisi oikea kulkuväylä pelikentän läpi. Aita-prefabin rikkominen hoitui lataamalla prefabin sprite-komponentin päälle uusi Resources/aitaRikki.png -tiedosto. Tämä muutos tapahtuu kameran taiton ulkopuolella pelaajan tietämättä.

5.5 Horisontti

Taustan kehitys pelin kokonaisuuden parantamiseksi on suuri osa pelin ilmapiirin luontia. Lapland Runnerin tausta koostuu Object Pool -tekniikkaa käyttävistä horisonttipalasista, taustanurmesta, horisontin taitosta sekä tulevista kenttäpalasista. Horisontin rakentaminen Object Pool -tekniikalla on käytännössä tehty kuten kentän generointi, mutta vain paljon pienemmässä mittakaavassa. Horisonttipaloihin on lisätty rigidbody-komponentti, jonka avulla työnnetään horisonttia vasemmalta oikealle x -akselilla ja liikutetaan sitä kameran mukana z-akselilla. Pelissä horisontin taittaminen tapahtui Unity Asset -kaupasta ostetulla assetilla, joka muokattiin pelin käyttöön sopivaksi (Andriyanov 2017). Tämän osan ulkoistaminen oli tarpeellista, koska projektissa ei riittänyt aika kaikkien asioiden kehitykseen.

5.6 Käyttöliittymä

Käyttöliittymän kehityksessä käytin ulkoista assettia nimeltä Unity UI extensions, tämän assetin avulla sain valikkorakenteelle hyvän pohjan (Jackson 2017). Käyttöliittymän rakentaminen assetilla oli helpompaa ja se vei vähemmän aikaa, kun se että olisi tehnyt kaiken itse. Työskentely assetin kanssa oli pääasiassa nappuloitten yhdistämistä, grafiikan asettamista oikeisiin paikkoihin ja pause- sekä voittoruudun kehitys.

6 TOTEUTUS

Projektin viimeisiä vaiheita on sen toteuttaminen. Toteuttaminen tapahtuu tekemällä pelistä versio Unity Editorin ulkopuolelle. Tätä versiota sanotaan koontiversioksi eli build-versioksi. Tärkeä osa oli myös katsoa, että peli näyttää koontiversiona samalta kuin sitä kehittäessä. Grafiikka tai muut osat voivat olla hyvin ongelmallisia pelin skaalatessa eri resoluutioihin. Pelille haettiin myös pelin aikana soivaa tekijänoikeusvapaata musiikkia.

6.1 Grafiikka

Graafinen piirtäminen oli pääasiassa tiimin graafikon tehtävä, mutta grafiikan lisääminen ja sovitteleminen peliin voi olla helppoa tai hyvinkin monimutkaista. Grafiikka asetettiin yleisesti peliobjektin päälle ja tarkastettiin, että grafiikka on talletettu oikeassa muodossa, jotta se veisi mahdollisimman vähän tilaa. Joskus kuvat muuttivat sävyjään Unityn tallentaessa ne Sprite (2D ja UI) -tilaan. Grafiikkaa tarvittiin pelin jokaiseen pelinäkymään, jotta peli saatiin näyttämään miellyttävältä (kuva 16).

KUVA 16. Pelin scenet

Pelin graafinen suunta on aivan yhtä tärkeä kuin pelimekaniikat ellei jopa tärkeämpi. Pelimekaniikoilla voidaan pitää kiinnostusta peliin vasta kun peliin on tutustuttu ja peligrafiikalla saadaan herätettyä pelaajalla mielenkiintoa ja kiinnostusta peliin heti ensi silmäyksellä.
Näyttävin osa grafiikkaa on yleisesti pelimaailma ja pelihahmo, mutta pelissä on myös paljon muuta pientä grafiikkaa kuten käyttöliittymän painikkeet, värit ja pienet ikonit.

6.2 Animointi

Pelin animointiin hyödynnettiin ulkoista SpriterDotnet -assettia. SpriterDotNet on asset, jonka avulla voi tuoda Spriter-sovelluksesta animaatiot Unity-projektiin (Sverko 2015). SpriterDotNet-asset sisältää kaikki tarvittavat skriptit animaation tekemiseen, jos animaatio tuodaan suoraa Spriter-sovelluksesta. Animaatiot tällä assetilla rakentuvat useasta eri palasesta kuten pelihahmon raajat, ja niiden pienien kuvien liikkeellä saadaan aikaiseksi elävä kokonaisuus. Pelihahmon eri ruumiinosien grafiikat liitetään pelihahmon prefabiin sen lapsiobjekteina, jonka avulla kääntöprefabit hallitsevat niiden kääntelyä.

Pelihahmon osuttua esteeseen sen väri vaihtuu hieman läpinäkyväksi, jonka avulla osoitetaan, että pelaajalla on estesuoja päällä. Pelihahmon värin vaihtaminen tapahtui yksinkertaisesti hakemalla kaikki SpriteRenderer-komponentit pelaajan lapsiobjekteista ja vaihtamalla sen läpinäkyvyysmuuttujan float arvon 1f -> 0.3f. Pelaajan estesuojan päättyessä värit palautetaan ennalleen. Ulkoisen assetin hyödyntämisen suurin syy on yksinkertaisesti ajansäästö. Animaatioiden tekeminen peliä varten vie paljon aikaa, jota projektissa ei ollut. Hyödyntämällä animointi assettia jää enemmän aikaa pelin kokonaisuuden parantamiseksi.

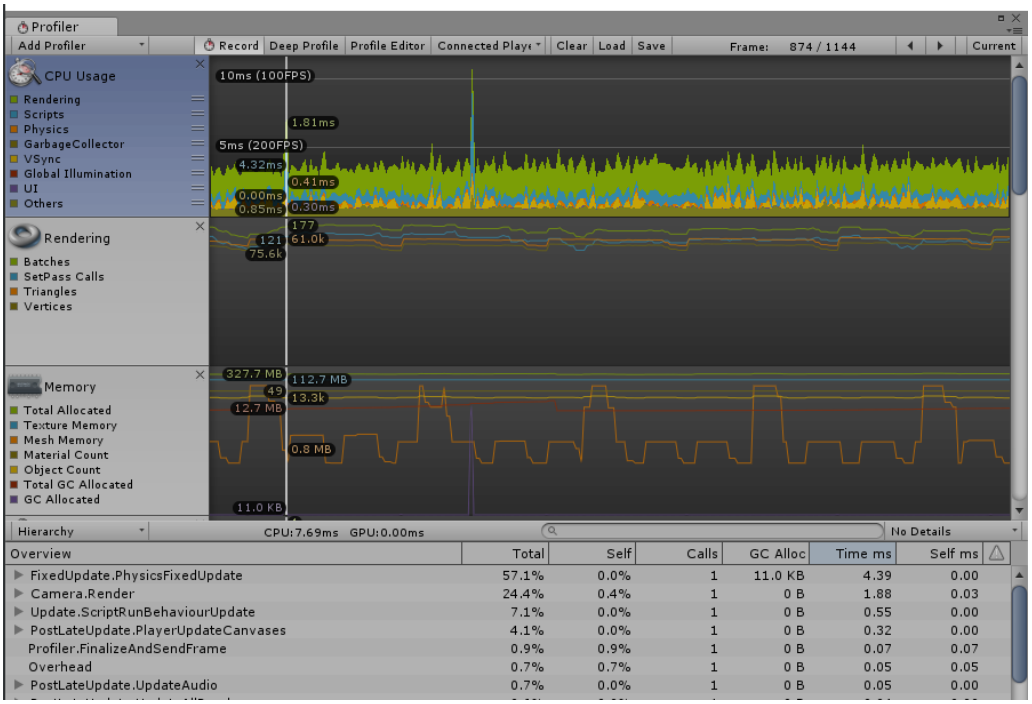
7 OPTIMOINTI

Pelin optimointi on moniosainen vaihe. Optimoinnissa kannattaa kiinnittää huomiota seuraaviin kohtiin:

- Skriptien optimoinnissa kannattaa puhdistaa skriptit kaikista ylimääräisistä riveistä ja funktioista. Skripteissä suurimmat pelin suorituskykyä hidastavat rivit ovat luupeissa tapahtuvat tulostukset, objektien luonnit ja tuhoamiset. Kaikki automaattisesti generoidut luokat voi skripteistä poistaa, jos niille ei ole tarvetta skriptin toiminnallisuuden kannalta.
- Grafiikan optimoinnissa on hyvä tarkistaa jokaisen erillisen kuvan kuvakoko. Suuri kuvakoko vaikuttaa build-version kokoon ja pelin graafiseen suorituskykyyn. Jos näkyvän grafiikan ei tarvitse olla tarkkaa tai se on liian kaukana, jotta sen voisi nähdä tarkasti, pienennä sen tarkkuutta porrastaen sopivaksi.
- Prefab-optimoinnissa on tärkeä varmistaa, että prefabeilla ole niille kuulumattomia komponentteja. Prefabien collider-komponentit on suotava pitää yksinkertaisia muotoonsa nähden.
- Audio-optimoinnissa on tärkeää kompressoida käytetyt ääni tiedostot niin pieneksi kuin on mahdollista. Kompressoi äänitiedostot porrastaen ja lopeta ennen kuin laatu laskee liikaa.

Tärkeää on myös varmistaa, ettei projektin Resource-kansiossa ole mitään ylimäärästä, koska tämä kansio ladataan koontiversion mukaan kokonaisuudessaan.

Unity Profiler on Unityn oma optimoinnin työkalu. Unity Profiler -työkalun avulla voidaan tarkastella resurssien käyttöä pelin aikana. Tämä työkalu tallentaa kaikkien resurssien käytön ja sen avulla voit tarkastella pelissä aiheutuvia suuria resurssipiikkejä, jotka aiheuttavat FPS:n eli kuvataajuuden putoamista (kuva 17).



KUVA 17. Unity Profiler

8 TESTAUS

Pelin testaus on iso osa pelin kehitystä. Testaamisella käydään läpi niin monta eri skenaariota pelin pelaamisesta kuin pystytään. Testauksen avulla pelimekaniikkoihin ja pelin kokonaisuuteen saadaan mahdollisimman virheetöntä toimintaa ja sen avulla ohjelmointivirheet huomataan jo kehitysvaiheessa. Pelin testausta tapahtuu koko projektin aikana, mutta suurin osa testausta suoritetaan yleensä pelin kehityksen loppuvaiheilla.

Testauksessa on myös tärkeää suorittaa pelitestausta sellaisten henkilöiden avulla, jotka eivät ole olleet pelin kehityksessä mukana. Testaus vaiheen aikana huomatut ohjelmointivirheet tai ei halutut toiminnallisuudet voidaan vielä korjata ennen kuin peli julkaistaan tai toimitetaan. Pelin testauksessa oli tärkeä varmistaa käyttöliittymän täydellinen toimivuus, tulosten tiedonkulku ja pelimekaniikkojen toimivuus. Testausta suoritettiin Unity Editor -ohjelmalla ja koontiversiona. Testausta suoritettiin yksinpelinä sekä moninpelina saadakseen mahdollisimman laajat testausvaihtoehdot.

9 POHDINTA

Opinnäytetyön tavoitteena oli tehdä valmis pelattava 1–4 tasapainolaudalla toimiva peli tilaajalle. Työ oli erittäin mielenkiintoinen tehdä, koska peli piti ensin ideoida, suunnitella, kehittää ja lopulta testata. Työn tilaajalla oli muutama vaatimus, kuten että peli toimii heidän kontrollereillaan, pelin teema olisi Lappi ja se käyttäisi Andreas Alarieston taidetta viitteenä. Kaikki tilaajan vaatimukset täytettiin.

Työssä kehitettävää oli erittäin paljon. Kaikista eri pelimekaniikoista tutkittiin mahdollisimman monta kehitysvaihtoehtoa ja kehitettiin parhaat sopivuudet peliä varten. Peliä kehittäessä on useita eri tapoja tehdä sama asia, esimerkiksi pelissä käytetty loputon pelikenttä. On tärkeä tutkia muita vaihtoehtoja parantaa pelimekaniikkaa ja pelin kokonaisuutta. Loputtoman pelikentän luonti oli yksi pelin suurimmista ominaisuuksista ja sen kehitykseen käytettiinkin lähestulkoon puolet projektille annetusta ajasta. Loputon pelikenttä käyttää Object Pool -tekniikkaa, jonka avulla pelin suorituskykyä paranneltiin kierrättämällä jo peliin ladattuja resursseja. Kierrättämällä pelin resursseja säästytään suurilta kuvantaajuuden putoamisilta, koska ei ole tarvetta luoda ja tuhota peliobjekteja. Isoja osia oli myös moninpeliominaisuudet ja testaus yhdessä asiakkaan laitteella. Testauksessa huomattiin monia asioita kuten että gyroanturi oli väärinpäin ja arvot tulisi invertoida, sekä muita ongelmia liittyen tietokoneen COM-portteihin.

Työn graafisen suunnittelun hoiti projektissa työskentelevä graafikko. Grafiikan implementointi peliin on mielenkiintoinen haaste, koska kaikki grafiikka pitää olla pienikokoista, jotta pelin koontiversiosta ei tulisi liian suurikokoinen. Pelin horisontin taitto ja animaatiot olivat myös mielenkiintoisia aiheita. Kehitin pelin animointia ja horisontin taittoa aluksi itse, mutta pienen selvityksen jälkeen löysin ulkoisia assetteja Unityn asset -kaupasta, jonka avulla ne saatiin integroitua peliin mahdollisimman vähällä työllä ajanpuutteen vuoksi. Pelin kehityksessä on tärkeä sisäistää sille annettu aika. Pelin kehitykseen voi käyttää vuosia saamatta sitä valmiiksi. Pitää huomioida, että tavoite oli saada valmis peli tietyssä ajassa.

Peli onnistui hyvin annettuun aikaan nähden ja se sisälsi kaikki tilaajan haluamat ominaisuudet. Jos pelin kehitykselle olisi annettu enemmän aikaa, olisin parantanut pelikentän monipuolisuutta lisäämällä uusia esteitä ja parantamalla pelikentän toteutus -skriptiä.

LÄHTEET

Andriyanov, Vadim 2017. Horizon Bending. Unity Asset Store. Viitattu 2.5.2019, https://assetstore.unity.com/packages/tools/particles-effects/horizon-bending-55306.

Gamedesigning 2018. Create Your First Game Design Document. Gamedesigning learn tutorials. Viitattu 28.5.2019, https://www.gamedesigning.org/learn/game-design-document/.

Geig, Mike 2014. Unity Technologies. Object Pooling in Unity. Unity Tutorials. Viitattu 4.4.2019, https://unity3d.com/learn/tutorials/topics/scripting/object-pooling.

Jackson, Samuell 2017. Unity UI Extensions community project. BitBucket Repository. Viitattu 3.5.2019, https://bitbucket.org/UnityUIExtensions/unity-ui-extensions/src/master/.

Paczkowski, Lukasz 2018. Replacing MonoDevelop-Unity with Visual Studio Community starting in Unity 2018.1. Unity Blog. Viitattu 24.5.2019, https://blogs.unity3d.com/2018/01/05/discontinuing-support-for-monodevelop-unity-starting-in-unity-2018-1/.

Sverko, Luka. 2015. A simple, fast and efficient Spriter implementation in pure C#. SpriterDotNet. Viitattu 4.5.2019, https://brashmonkey.com/forum/index.php?/topic/4166-spriterdotnet-an-implementation-for-all-c-frameworks/.

Unity Technologies 2019a. Order of execution for event functions. Unity Manual. Viitattu 2.5.2019, https://docs.unity3d.com/Manual/ExecutionOrder.html.

Unity Technologies 2019b. Object.DontDestroyOnLoad. Scripting API. Unity Documentation. Viitattu 2.5.2019, https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html.

Unity Technologies 2019c. Object.Instantiate. Scripting API. Unity Documentation. Viitattu 2.5.2019, https://docs.unity3d.com/ScriptReference/Object.Instantiate.html.

LIITTEET

Liite 1 ObjectRearrangeScript.cs

Liite 2 ObjectPoolScript.cs

Liite 3 EndlessPoolerScript.cs

Liite 4 EndlessPoolerDestroyScript.cs

OBJETREARRANGESCRIPT.CS

LIITE 1

```
public class ObstacleRearrangeScript : MonoBehaviour {
    int mapBorder = 5;

    Transform myChildObstacle;

    void OnDisable()
    {
        //haetaan testSetin alta viiminen objecti vasta kun esteet ovat syntyneet ja liikutellaan sitä joka disablessa.
        if(transform.childCount >= 35)
        {

            myChildObstacle = transform.GetChild(34);
            int spawnPointX = Random.Range(-mapBorder, mapBorder);
            myChildObstacle.transform.position = new Vector3(spawnPointX, transform.position.y, transform.position.z);

        }

    }
}
```

```
public class ObjectPoolScript : MonoBehaviour
{
    public static ObjectPoolScript current;
    public GameObject[] pooledObject;
    public int pooledAmount = 5;
    public bool willGrow = true;
    public Object[] obstaclePrefabs;


    public List<GameObject> pooledObjects;

    void Awake()
    {
        current = this;
    }

    void Start()
    {
        obstaclePrefabs = Resources.LoadAll("Prefabs");
        pooledObjects = new List<GameObject>();
        for (int i = 0; i < pooledAmount; i++)
        {
            for (int x = 0; x < pooledObject.Length; x++)
            {
                int iRandomIndex = Random.Range(0, pooledObject.Length);
                GameObject obj = (GameObject)Instantiate(pooledObject[iRandomIndex]);
                obj.SetActive(false);
                pooledObjects.Add(obj);
                GetPooledObject();

            }
        }
    }

    public GameObject GetPooledObject()
    {
        for (int i = 0; i < pooledObjects.Count; i++)
        {
            for (int x = 0; x < pooledObject.Length; x++)
            {

                if (pooledObjects[i] == null)
                {
                    //Debug.Log("GetPooledObject");
                    GameObject obj = (GameObject)Instantiate(pooledObject[x]);
                    obj.SetActive(false);
                    pooledObjects[i] = obj;
                    return pooledObjects[i];
                }
                if (!pooledObjects[i].activeInHierarchy)
                {
                    return pooledObjects[i];
                }
            }
        }

        if (willGrow)
        {
            GameObject obj = (GameObject)Instantiate(pooledObject[0]);
            pooledObjects.Add(obj);
            return obj;
        }

        return null;
    }

}
```

ENDLESSPOOLERSCRIPT.CS


```
public class EndlessPoolerScript : MonoBehaviour
{
    //GameManagerScript
    public GameManagerScript gameManagerScript;

    public Vector3 newPos;
    GameObject[] players;
    GameObject startPiece;
    GameObject[] groundPieces;
    public float moveBackPos;
    int i =  0;
    void Start() {
        startPiece = GameObject.FindGameObjectWithTag("StartPiece");
        gameManagerScript = GameObject.FindGameObjectWithTag("GameManager").GetComponent<GameManagerScript>();

        Debug.Log("Invoke Endlesspooler");
        InvokeRepeating("Spawn", 5, 0.4441f);
        //0.4441f

    }

    void Spawn()
    {
        GameObject obj = ObjectPoolScript.current.GetPooledObject();

        if (obj == null) return;

        groundPieces = GameObject.FindGameObjectsWithTag("GroundPieces");
        obj.transform.position = newPos;
        newPos = newPos + (transform.forward * 4);

        //Move platforms back -- ResetWorldPosition
        if(newPos.z > (moveBackPos + 60))
        {

            newPos = new Vector3(0, 0, 60);
            gameManagerScript.ResetPlayerWorldPos();

            foreach (GameObject gp in groundPieces)
            {
                i++;
                gp.transform.position = gp.transform.position + (transform.forward * -moveBackPos) + (transform.forward * 4);
                Debug.Log(i++);
                Debug.Log(gp.transform.position);

            }

        }

        obj.SetActive(true);

    }

    IEnumerator WaitObjectPooling()
    {
        yield return new WaitForSeconds(5);
        Debug.Log("WaitObjectPooling");
    }
}
```

ENDLESSPOOLERDESTROYSCRIPT.CS LIITE 4

```
public class EndlessPoolerDestroyScript : MonoBehaviour
{

    void OnEnable()
    {
        Invoke("Destroy", 10f);
    }

    void Destroy()
    {
        gameObject.SetActive(false);
        //Debug.Log("Destroy Script");
    }

    void OnDisable()
    {
        CancelInvoke();
    }
}
```