OULUN AMMATTIKORKEAKOULU

Ville Vaara

ORTOGRAFINEN PROJEKTIO & SYVYYSVAIKUTELMA

Kaksiulotteisen pelin kehitys Unity-pelimoottorilla

OULUN AMMATTIKORKEAKOULU

ORTOGRAFINEN PROJEKTIO & SYVYYSVAIKUTELMA

kaksiulotteisen pelin kehitys Unity-pelimoottorilla

Ville Vaara
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, Ohjelmistokehitys

Tekijä(t): Ville Vaara
Opinnäytetyön nimi: Ortografinen projektio & syvyysvaikutelma
Työn ohjaaja(t): Janne Kumpuoja
Työn valmistumislukukausi ja -vuosi: Kevät 2024 Sivumäärä: 28

Opinnäytetyön aiheena oli oppimismielessä tutkia pelinkehitystä Unity-pelimoottorilla, sekä toteuttaa opitun perusteella kaksiulotteisen tasohyppelypelin tekninen demo. Aluksi tarkasteltiin kaksiulotteisen pelikehitystä yleisellä tasolla, minkä jälkeen erityistarkasteluun otettiin syvyysvaikutelman luomisen keinot ortografisessa näkymässä. Työn lopputuloksena oli toimiva tekninen demo mahdollista jatkojalostusta varten.

Asiasanat: Unity, peliala, peliohjelmointi, peligrafiikka, perspektiivi, projektio, mittasuhteet

ABSTRACT

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

---

Author(s): Ville Vaara
Title of thesis: Ortographic projection & sense of depth
Supervisor(s): Janne Kumpuoja
Term and year when the thesis was submitted: Spring 2024
Number of pages: 28

The topic of this thesis was to research process of making games on Unity game engine and to produce a technical demo of a 2d platformer. At first, we look at developing games on Unity at more general sense, after which we focus on creating a sense of depth in an orthographic projection. As result of this thesis was a functional technical demo ready to be used as a base for further development.

Keywords: Unity, game sector, game programming, game graphics, perspective, projection, proportions

SISÄLLYS

1 JOHDANTO

Opinnäytetyön tarkoituksena on tutkia kaksiulotteisen pikselipohjaisen tasohyppelypelin kehitystä Unity-pelimoottorilla, sekä toteuttaa opitun kautta tekninen demo jatkokehitystä varten. Alustuksena käydään läpi pelimoottorin toimintaa yleisellä tasolla, sekä tutustutaan sen tarjoamiin keskeisiin toiminnallisuuksiin kaksiulotteisten pelien kehittämiseen.

Produktiossa kiinnitetään erityistä huomiota resurssiteknisiin seikkoihin johtuen aikataulutuksesta sekä projektin käytössä olevista rajallisista henkilöresursseista. Tämän realiteetin sanelemana graafiseksi toteutustavaksi valikoitui rajatun resoluution pikselitoteutus, jossa painopisteenä on tehokas grafiikan luonti sekä luotujen elementtien uudelleenkäytettävyys. Pikselipohjaisen toteutuksen vuoksi kameran projektioksi valittiin ortografinen kamera, jonka kautta löytyi myös opinnäytetyön erityistarkastelun kohde: Kuinka kohtisuorassa projektiossa luodaan syvyysvaikutelma?

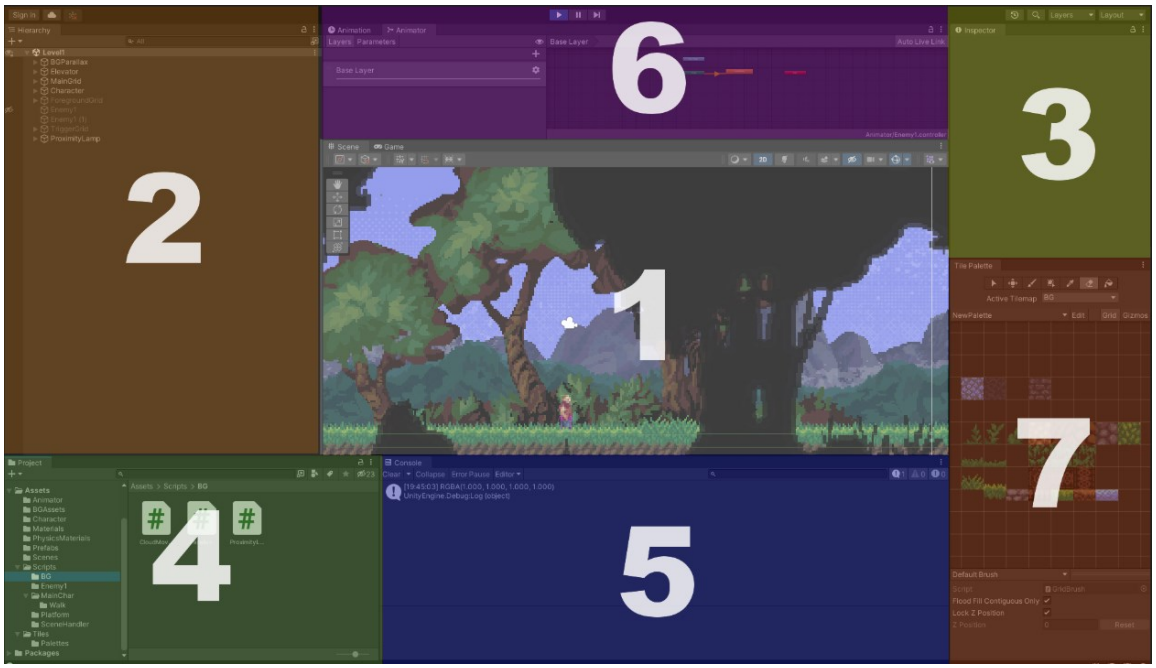
Lopuksi tarkastellaan, miten opinnäytetyön produktio-osuus eli tekninen demo toteutettiin. Pohdinnassa mietitään pikselipohjaisen toteutuksen kaupallista potentiaalia.

2 KAKSIULOTTEISEN PELIN KEHITYS UNITYLLÄ

2.1 Käyttöliittymä lyhyesti

Ohessa kuvattuna opinnäytetyön produktion toteutukselle oleelliset Unityn UI-elementit sekä niiden selitteet (kuva 1):

1. Näkymät: Tärkeimpinä näkyminä pelinäkymä sekä skenenäkymä. Pelinäkymä vastaa pelin aktiivisen kameran piirtämää aluetta, eli sitä miten peli näyttäytyy loppukäyttäjälle. Skenenäkymää puolestaan käytetään kehitystyössä visualisoimaan esimerkiksi törmäyttimien kehyksiä, kameran projektion piirtokehikoita sekä muita kehitystyössä oleellisia teknisiä seikkoja.
2. Hierarkiavalikko: Listaa tällä hetkellä ladatut skenet sekä kaikki niiden sisältämät peliobjektit. valikon kautta voi hallinnoida elementtejä tai muuttaa niiden keskinäistä suhdetta hierarkiassa.
3. Inspektori: Käytetään valittuna olevan peliobjektin ominaisuuksien muokkaamiseen tai uusien komponenttien lisäämiseen. Komponentit ovat peliobjektien ominaisuuksia laajentavia toiminnallisuuksia tai skriptejä.
4. Projektin tiedostot ja kansiot, sisältäen projektin resurssit, kuten kuvat, äänet, animaatiot, materiaalit, skriptit tai esimerkiksi edellä mainituista valmiiksi kootut uudelleenkäytettävät kokonaisuudet eli prefabit.
5. Konsoli: Näyttää pelin ajonaikaiset virheet, varoitukset, sekä skripteistä lokitetut debug-viestit. Debug-konsoli on jokaiselle ohjelmointia jollain tasolla toteuttaneelle tuttu työkalu.
6. Animaattori: Toteutetaan elementtien animaatiot sekä siirtymien logiikka.
7. Tilemap-paletti. Käytetään laattapohjaisessa toteutuksessa pelikentän ”maalaamiseen” ruudukko -tyyppiseen peliobjektiin skenen hierarkiassa. (1.)



KUVA 1. Unityn tärkeimmät UI-elementit.

2.2 Skenet, peliobjektit sekä komponentit

Unityssä peliprojekti koostuu yhdestä tai useammasta skenestä. Useasti yksi skene edustaa yhtä pelikenttää (2). Kerralla voi kuitenkin olla ladattuna myös useampia skenejä, joita voidaan myös ladata tai poistaa ajonaikaisesti. Ajonaikainen dynaaminen aktiivisten skenejen hallinnointi auttaa optimoimaan suorituskykyä laajemmissa kenttäkokonaisuuksissa. Näin voidaan välttää myös kankeita latausruutupohjaisia siirtymiä. (3.)

Jokaisessa skenessä peruselementtinä on ”GameObject” eli vapaasti suomennettuna peliobjekti. Käytännössä kaikki loppukäyttäjälle näyttäytyvät pelin toiminnalliset elementit, kamera mukaan lukien, ovat peliobjekteja. Peliobjekteihin voidaan myös liittää erilaisia komponentteja toteutuksen tarpeiden mukaisesti. Komponentit ovat peliobjektien ominaisuuksia laajentavia toiminnallisuuksia tai skriptejä. (4.)

Peliobjektit ovat hierarkiassa keskenään joko rinnakkaisia, tai toistensa isäntä- tai lapsielementtejä. Lapsielementit perivät isäntäelementiltä ominaisuuksia, kuten esimerkiksi juurikoordinaatit. Tällä on inkrementaalinen vaikutus, mikäli myös lapsielementin omia koordinaatteja muokataan. Lapsielementin koordinaatit eivät siten ole absoluuttisia, vaan suhteellisia isäntäelementin juurikoordinaatteihin. Tämän vuoksi ne voidaan mieltää offset-arvoiksi. (5.)

2.3 Pelin fysiikka

Kaksiulotteisissa toteutuksissa ei kolmiulotteisista toteutuksista poiketen yleensä ole tarvetta käsitellä syvyyssuunnan akselia pelin fysiikan osalta, sillä kaikki ydintoiminnot tapahtuvat vaaka- ja pystyakseleilla. Tämä on huomioitu myös Unityn fysiikkamoottorin tarjoamissa komponenteissa. Eri komponenteille löytyvät omat kaksiulotteiset vastineensa, jotka käsittelevät dataa vain kahdella akselilla. Käsiteltävät arvot voidaan siten ilmaista kaksiulotteisena vektorina eli vector2-tyyppisinä tietueina. (6.)

Jotta fysiikkamoottorilla voidaan vaikuttaa peliobjektiin, täytyy siihen olla liitettynä Rigidbody-komponentti. Rigidbody2D eli kaksiulotteinen ”jäykkä kappale” on kontrolleri elementille, jonka kanssa muut fysiikkamoottoria käyttävät elementit voivat olla vuorovaikutuksessa. Kappale voi olla joko staattinen tai dynaaminen, mutta yhtä kaikki, mikäli elementin halutaan olevan pelissä ”fyysinen”, tulee sillä olla tällainen kontrolleri. (7.)

Rigidbodyn lisäksi tarvitaan Collider eli törmäytin. Törmäytin on peliobjektiin liitettävä komponentti, joka määrittelee kyseisen objektin muodon fysikkamoottorille. Törmäytin tarkkailee kyseisen elementin kontaktia toisten elementtien törmäyttimien kanssa. Tätä voidaan hyödyntää fyysisten törmäysten lisäksi esimerkiksi erilaisten triggereiden laukaisemisessa (8). Törmäyttimelle voidaan määritellä myös ”PhysicsMaterial”, jolla määritetään, miten muut elementit reagoivat törmäykseen kappaleen kanssa. Määritettävissä on esimerkiksi materiaalin kitka sekä kimmoisuus. (9.)

Eri käyttötarkoituksiin soveltuvia törmäyttimiä on runsaasti, mutta niiden perustoiminto on muutamaa erityistapausta lukuun ottamatta sama. Merkittävin eroavaisuus useimpien törmäyttimien välillä on niiden muodon määrittelevän kehikon kompleksisuus. Yksinkertaisimmillaan törmäytin on primitiivi muoto, kuten BoxCollider2d, joka on nimensä mukaisesti neliskanttinen törmäytin. Se on helppokäyttöisyytensä vuoksi yksi kaikkein yleisimmin käytetyistä törmäyttimistä. Toimiakseen sille määritellään ainoastaan törmäyttimen koko vaaka- ja pystyakselilla, sekä tarvittaessa akseleiden offset-arvot.

Erityisten käyttötapauksiensa vuoksi mainitsemisen arvoisia törmäyttimiä ovat TilemapCollider2D sekä CompositeCollider2D. TilemapCollider2D on tarkoitettu käytettäväksi laattapohjaisessa pelikenttätoteutuksessa. Jokaiselle ruudukon laatalle piirretään oma törmäyttimen kehikko riippuen siitä, miten kehikko on kullekin laatalle määritelty ”sprite editor” -työkalussa. (10.)

CompositeCollider poikkeaa toiminnoltaan muista törmäyttimistä. Aiemmin mainituista törmäyttimistä poiketen sitä ei käytetä itsenäisenä törmäyttimenä, vaan sen funktio on yhdistellä eri törmäyttimien kehikoita siten, että niistä muodostuu yksi yhtenäinen alue. CompositeCollider asetetaan hierarkiassa komponentiksi isäntäelementtiin, jolloin lapsielementtien törmäyttimet yhdistellään yhdeksi yhtenäiseksi kehikoksi, jos lapsielementin törmäyttimen ominaisuuksissa on valittuna ”UsedByComposite”. CompositeCollider on erityisen tärkeä laattapohjaisessa toteutuksessa, eli sitä käytetään usein yhdessä TilemapCollider2d:n kanssa, sillä se poistaa turhia törmäyttimen kehikon osia jättäen jälkeensä eheän ja yhtenäisen törmäyttimen (kuva 2). Tämä vähentää pelin fysiikan potentiaalista ei-toivottua käyttäytymistä sekä optimoi suorituskykyä. (11.)

Tapahtuneen törmäyksen tunnistamisen lisäksi joskus on myös tarpeen tunnistaa jossain suunnassa vastaan tuleva törmäytin jo ennen kontaktia toisen törmäyttimen kanssa. Tähän tarkoitukseen soveltuvia työkaluja ovat erilaiset luotausmenetelmät, kuten esimerkiksi RayCast, joka luotaa haluttuun suuntaan säteen avulla (12). Toisena vaihtoehtona on BoxCast, joka puolestaan luotaa neliskulmaisella kehikolla (13). Näillä luotausmenetelmillä pystytään havaitsemaan halutulta etäisyydeltä ensimmäinen vastantuleva törmäytin.

KUVA 2. TilemapCollider2D-elementin asetus "Used By Composite" poistaa ylimääräiset törmäyttimen kehikon osat laattojen välistä.

2.4 Koodin rakenne ja skriptien tuottaminen

Unityssä skriptien ohjelmointikielenä on lähtökohtaisesti C#. Muuttujien, objektien sekä funktioiden näkyvyys toimii kuten ohjelmointikielissä normaalisti. Näkyvyydet ovat Public, Private sekä Protected, eli Julkinen, Yksityinen ja Suojattu. Mikäli perivän luokan halutaan pystyvän käsittelemään muuttujaa, määritellään se suojatuksi. Toisaalta jos muuttujaa, objektia tai funktiota pitää pystyä käsittelemään tai kutsumaan skriptin ulkopuolelta, haluttu näkyvyys on julkinen. Julkisen muuttujan arvoa tai julkiseksi määriteltyä peliobjektiviitettä pystyy muuttamaan koodiin kajoamatta suoraan Unityssä sen peliobjektin alta, johon skripti on liitettynä. Muissa tapauksissa hyvän ohjelmointitavan mukaisesti näkyvyys määritellään yksityiseksi, eli suppeimmalla tarpeet täyttävällä näkyvyydellä. (14.)

Koodin tuottamista Unityssä helpottaa suuri määrä valmista toiminnallisuutta sekä erinomainen dokumentaatio. Hallitsevan markkina-asemansa vuoksi Unityllä on myös laaja verkkoyhteisö, josta aloitteleva pelinkehittäjä voi löytää vertaistukea ongelmatilanteissa.

UnityEngine.CoreModule sisältää useimpien skriptien kantaluokkana toimivan ”Monobehavior”-luokan. Kantaluokan alla on runsaasti valmiiksi määriteltyjä funktioita kehitystyön sujuvoittamiseksi. Niistä tärkeimmät ovat seuraavat ajonaikana kutsuttavat funktiot:

- Start(): Kutsutaan kerran, kun peliobjekti on luotu ja siihen liittyvä skripti on käynnistetty. Objektin alustukseen liittyvät asiat käsitellään tässä.
- Update(): Kutsutaan kerran jokaisella ruudunpäivityksellä. Tässä funktiossa tapahtuvat yleensä mm. pelaajan liikkeet ja kameran hallinta.
- LateUpdate(): Kutsutaan heti Update()-funktion suorittamisen jälkeen.
- FixedUpdate(): Kutsutaan säännöllisin väliajoin. Tämä funktio on suunniteltu käsittelemään esimerkiksi pelin fysiikkaa, sillä se ei ole ruudunpäivityksen kuvataajuuteen sidottu kuten Update-funktio. Eli toisin sanoen ruudunpäivitystaajuuden heittelyt eivät vaikuta funktion toimintaan.
- OnEnable(): Kutsutaan, kun peliobjekti otetaan käyttöön.
- OnDisable(): Kutsutaan, kun peliobjekti poistetaan käytöstä.

Lisäksi triggereitä sekä törmäyksiä varten löytyy myös valmiit tapahtumapohjaiset kutsunsa. (15.)

Skriptien tuottamisen näkökulmasta ehdottomasti mainitsemisen arvoisia ovat lähes joka projektissa käytettävät Time-luokan määrittämät muuttujat DeltaTime sekä FixedDeltaTime. DeltaTime on intervalli kahden ruudunpäivityksen välillä, eli se on verrattavissa Update()-kutsun intervalliin. FixedDeltaTime on ruudunpäivitystaajuudesta riippumaton intervalli, eli se vastaa FixedUpdate()-kutsun intervallia. (16.)

2.5 Projektin graafiset elementit

2.5.1 Graafisten resurssien tuonti

Kun graafisia elementtejä tuodaan Unityyn, on huomioitava muutamia teknisiä seikkoja, kuten käytettävä polygoniverkon tyyppi, kuvan formaatti, sekä kuvan suodatus mäpättäessä tekselille.

Peliprojektin kaksiulotteista kuvaresurssia kutsutaan nimellä sprite. Sprite toimii Unityssä kuten tekstuurit toimivat 3D-toteutuksessa: Sprite mäpätään tekstuuriksi polygoniverkolle. Polygoniverkon tyypiksi voidaan valita ”Full Rect”, joka nimensä mukaisesti on neliskanttinen alue, joka sul-

kee sisäänsä spriten kokonaisuutena, mukaan lukien mahdolliset ”tyhjät” eli läpinäkyvät alueet. Vaihtoehtoisesti valinta voi olla ”Tight”, eli tiukka rajaus, jossa tyhjät alueet jätetään polygoniverkon ulkopuolelle riippuen kunkin pikselin alfakanavan arvosta. Tiukalla rajauksella polygoniverkkoon tulee monimutkaisempi rakenne, mutta se on yleensä parempi valinta suorituskyvyn näkökulmasta, koska tällöin läpinäkyvät pikselit jätetään käsittelemättä. On kuitenkin myös tilanteita jolloin ”Full Rect” on oikea valinta. Esimerkkinä tästä kuvaresurssi, jota halutaan toistaa yhdellä tai molemmilla akseleilla. Tämänkaltaisissa tapauksissa Unity antaa käyttäjälle kehotteen vaihtaa polygoniverkon tyypiksi Full Rect, jotta kuva toistuu oikein. (17.)

Kuvan formaatin valintaa varten tulee tietää tarvittava väriavaruus, sekä käytössä olevat värikanavat. Esimerkiksi RGB-formaatissa käytössä on 3 värikanavaa, eli punainen, vihreä ja sininen. RBGA-formaatti puolestaan tukee edellisten lisäksi myös Alfakanavaa, jolla yleensä määritellään piirrettävän tason peittävyys. Väärää formaattia ei ole, vaan valinta riippuu käyttötarpeesta. Lisäksi graafista resurssia tuodessa tulee aina ottaa huomioon kuvan suodatus, mutta erityisen tärkeää se on matalan resoluution toteutuksissa. Valittu suodatus määrittää sen, miltä kuvan pikseli näyttää tekstuurikartan kuviointialkiolla eli tekselillä. Useimmille ensikosketus erilaisiin suodatuksessa käytettyihin interpolaatiomenetelmiin tulee kuvankäsittelyohjelmista kuvien skaalauksen yhteydessä: kun kuvaa skaalataan, millä logiikalla täytetään puuttuva data? Myös pikselin mäppäys tekselille Unityssä toimii samanlaisella logiikalla. (18.)

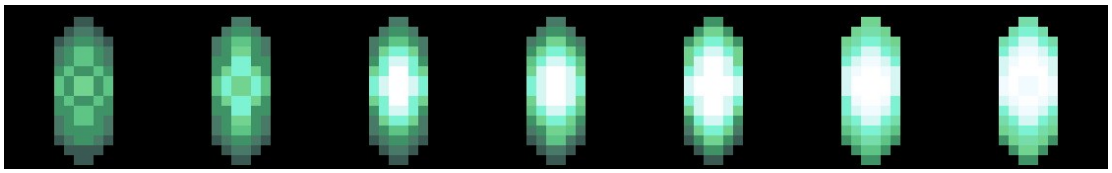
Pikseligrafiikalle haluttu suodatus on käytännössä aina pistesuodatus, jossa kuvan pikseli kartoitetaan tekselille lähimmän naapurin interpolointimenetelmää käyttäen (19). Näin pikseligrafiikan reunat pysyvät terävinä ilman sumennusta. Bilineaariseen tai trilineaariseen interpolaatioon perustuvat filtterit puolestaan laskevat painotetun keskiarvon tekselille sitä ympäröivästä pikseliryhmästä. Tämä voi riittävän suuriresoluutioisen kuvan osalta olla haluttu lopputulema, mutta se saa matalan resoluution kuvan vaikuttamaan sumealta tai pikseligrafiikan kohdalla täysin käyttökelvottomalta (kuva 3). (20.)

KUVA 3. Järjestyksessä vasemmalta oikealle: pistefiltteri (nearest neighbor), bilineaarinen filtteri, trilineaarinen filtteri

2.5.2 Spritesheet ja piirtokutsujen optimointi

Unity kutsuu jokaista skenessä käytettävää kuvaresurssia erikseen. Sprite-grafiikan tarvelähtöisenä ideana on koostaa useampi graafinen resurssi yhteen tiedostoon, eli ”spritesheet”-kuvaresurssiin (kuva 4). Yhdessä kuvaresurssissa on tyypillisesti kaikki animaation ruudut tai esimerkiksi kaikki pelikentän laatat. Jotta kuvaresurssista osataan noutaa yksittäinen ruutu tai laatta oikean kokoisena, tulee solun koko määritellä SpriteEditor-työkalussa. Yksittäisille laatoille voidaan määritellä samassa yhteydessä törmäyttimen kehikko. (21.)

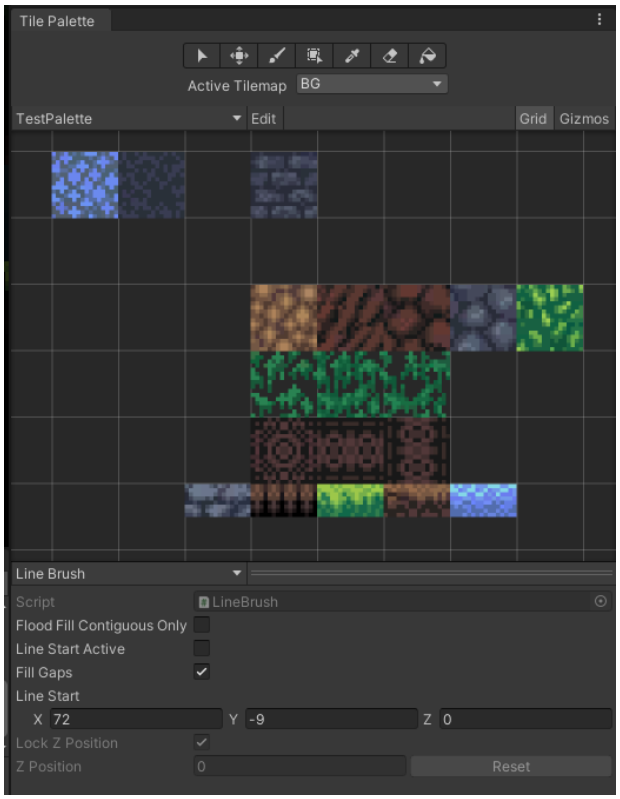
Spritesheet vähentää kuvaresurssien piirtokutsuja eli optimoi laiteresursseja. Mikäli piirtokutsuja on edelleen suorituskyvyn kannalta liikaa, voivat ne aiheuttaa piirtovirheitä tai hitautta. Kuvaresurssien kutsujen optimoimisessa voidaan hyödyntää Sprite Atlas -työkalua, joka yhdistelee useampia resursseja yhdeksi isommaksi kirjastoksi. Yksittäisten resurssien kutsumisen sijaan Unity kutsuu suoraan Sprite Atlaksen luomaa kirjastoa. (22.)



KUVA 4. Lamppu. Spritesheet-esimerkki. Animaation kaikki ruudut tuodaan Unityyn yhtenä kuvana

2.5.3 TilePalette ja Grid

Laattapohjainen kenttätoteutus on suosittu kaksiulotteisissa peliprojekteissa ilmeisestä syystä. Laatat ovat uudelleenkäytettäviä resursseja, eli verraten pienellä työmäärällä saadaan aikaiseksi paletti, jolla voidaan rakentaa laajoja kenttäkokonaisuuksia. Kuvaresurssin laatat tuodaan TilePalette-työkaluun (kuva 5). (23.) Paletin kautta ne voidaan maalata suoraan skeneen lisättyyn ruudukkoon eli ”Grid”-tyyppiseen peliobjektiin (24). Kuvaresurssia voidaan myös jälkikäteen päivittää, jolloin tehdyt muutokset päivittyvät suoraan jo valmiiksi rakennettuihin kenttiin.



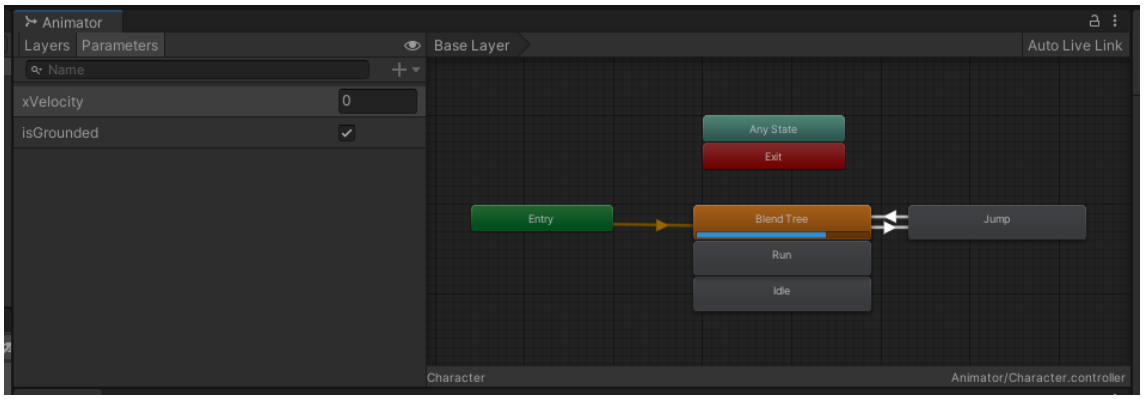
KUVA 5. TilePalette

2.5.4 Sprite-maskit

Sprite-maskeilla määritellään, mikä osa grafiikasta on näkyvissä. Grafiikkaelementin ”SpriteRenderer”-komponentista voidaan valita, miten kukin elementti reagoi maskiin. Mikäli kyseisen peliobjektin halutaan reagoivan tiettyyn maskiin, on vaihtoehtoina grafiikan näkyminen vain maskin sisäpuolella tai vain sen ulkopuolella. Maskit ovat monikäyttöisiä työkaluja ja hyviä käyttötapauksia on monia. Parhaimmillaan niitä hyödyntämällä säästetään henkilöstöresursseja grafiikan tuottamisessa tai niitä hyödyntäen voidaan rakentaa peliin kiinnostavaa pelimekaniikkaa. (25.)

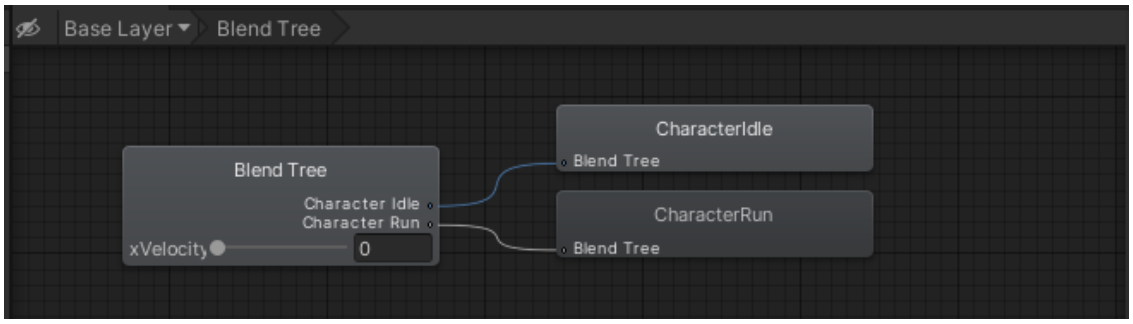
2.5.5 Animaatioiden hallintaelementti ja tilakone.

Yksittäinen animaatioresurssi luodaan spritesheetin pohjalta Animator Timeline -työkalussa. Työkalun aikajanalla voidaan hallinnoida yksittäisiä ruutuja ja niiden ajastuksia. Unityn animaatioiden hallintaelementti eli ”Animator controller” on työkalu, jonka funktio on vaihtaa eri animaatioresurssien välillä ennalta määritettyjen ehtojen täyttyessä. Tätä mekanismia kutsutaan tilakoneeksi (kuva 6). (26.)



KUVA 6. Spriten animoinnissa voidaan hyödyntää Unityn tilakoneseen pohjautuvaa animaatiotyökalua.

Tilan eli näytettävän animaation voi vaihtaa joko heti ehdon täytyttyä tai hallitummin vaiheittaisen siirtymän kautta. BlendTree-elementti auttaa vaihtamaan animaatiota hallitummin kahden tai useamman animaation välillä riippuen määritellyistä raja-arvoista, halutuista siirtymän tyypeistä ja ajastuksesta. BlendTree mahdollistaa myös asteittaiset siirtymät eri animaatioiden välillä (kuva 7). (27.)

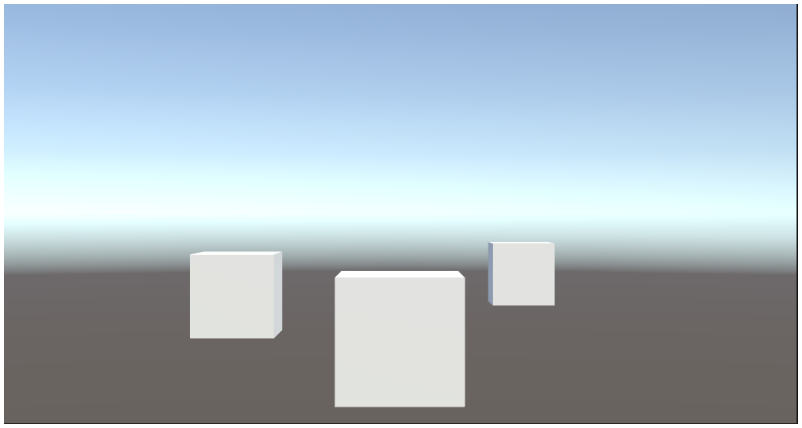


KUVA 7. Animaattorin blendtree vaihtaa animaatiota perustuen skriptissä määritellyn raja-arvon mukaan. Tässä tapauksessa muuttuja on nimetty nimellä xVelocity.

3 SYVYYSVAIKUTELMAN LUONTI ORTOGRAFISESSA PROJEKTIOSSA

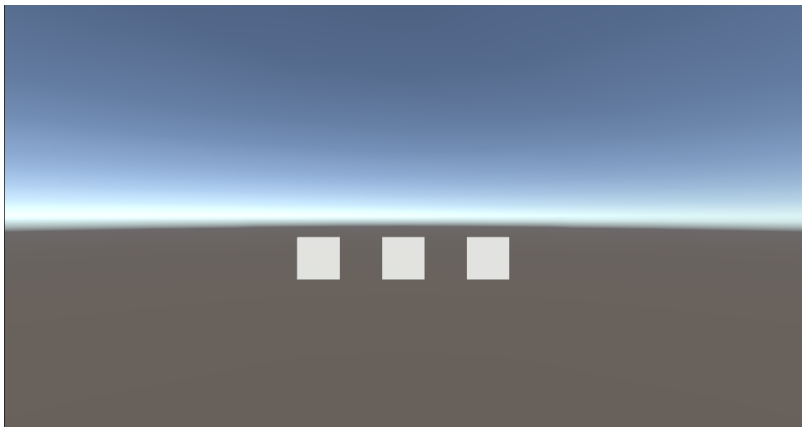
3.1 Kameraprojektiot

Perspektiivikamera luo realistisen kolmiulotteisen tilan vaikutelman johtuen projektion luonteesta. Näkymällä on ”pakopiste”, jota kohti mentäessä objektit suhteessa kutistuvat. Mitä kauempana kamerasta elementti sijaitsee, sitä pienempänä se näyttäytyy katsojalle (kuva 8). Elementit myös liikkuvat perspektiivin mukaisesti eli kauempana olevat elementit liikkuvat hitaammin suhteessa kameranäkymään. Syvyysvaikutelma syntyy automaattisesti. Näkymä toimii siis ihmissilmän tavoin.



KUVA 8. Perspektiivikamera, Kappaleiden XYZ -koordinaatit: -2,0,10 / 0,0,5 / 2,0,15

Ortografisessa kamerassa puolestaan kaikki kameran syvyysalueella olevista elementeistä piirtyvät kameralle ”oikeassa koossaan” kohtisuorana projektiona ilman minkäänlaista vääristymää (kuva 9). Tämä tarkoittaa sitä, ettei syvyysvaikutelmaa synny luonnostaan projektion vääristymän kautta kuten perspektiivisessä näkymässä, vaan pitää se luoda muilla tavoin. Ortografinen kamera on oikea valinta silloin, kun halutaan graafisten elementtien mittasuhteiden pysyvän samoina. Yleensä se tulee kyseeseen, kun puhutaan kaksiulotteisista toteutuksista. Ortografisen kameran merkitys korostuu silloin, kun käytetään rajoitettua resoluutiota. Mikäli toteutuksessa on päädytty pikseligrafiikkaan, yleensä haluttu lopputulos on se, että pikselin koko pysyy vakiona. (28).



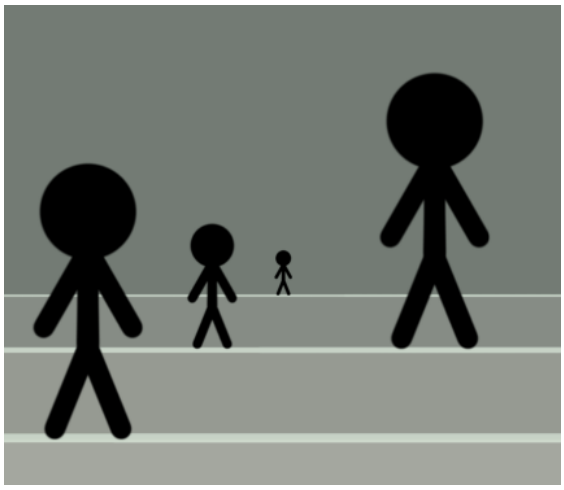
KUVA 9. Ortografinen kamera. Kuvassa näkyvien kappaleiden XYZ-koordinaatit: -2,0,10 / 0,0,5 / 2,0,15

3.2 Syvyysvaikutelma ortografisella kameralla.

3.2.1 Parallaksi

Ortografinen projektio on kohtisuora projektio. Koska projektiossa ei ole vääristymää, ei siinä ole myöskään luonnollista syvyysvaikutelmaa. Siksi syvyysvaikutelma on luotava visuaalisin tehokeinoin. Yksi näistä keinoista on parallaksi.

Kaksiulotteisissa graafisissa toteutuksissa parallaksiefektillä tarkoitetaan liikkuvilla tasoilla emuloitua perspektiivi-illuusiota. Illuusio luodaan siten, että tason liikkeen nopeus on sidottu sen näennäiseen etäisyyteen kamerasta. Lähempänä kameraa olevat tasot liikkuvat nopeammin kuin kauempana olevat tasot. Tasojen liikkeen nopeus korostaa myös tasojen näennäistä skaalaa. Menetelmällä pyritään imitoimaan sitä, miten kappaleet liikkuvat perspektiiviprojektiossa. Perspektiivikamerassa elementit pienenevät pakopistettä kohti mentäessä, eli toisin sanoen perspektiivikameran näkymän kehikko laajenee mitä etäämmälle kamerasta mennään. (Kuva 10) Kauemmat tasot kattavat siten huomattavasti laajemman pinta-alan, eli pysyvät liikkuessa näkökentässä pidempään. Katsojan näkökulmasta kauempana olevat elementit siis todella liikkuvat hitaammin.



KUVA 10. Etuala, keskiala ja taka-ala yksinkertaistetussa perspektiivissä. Keskialan suurempi hahmo on kopio etualan hahmosta, vaikka visuaalisesti keskialan hahmo vaikuttaa suuremmalta kontekstin vuoksi.

Parallaksitasoja voidaan käyttää haluttu määrä, mutta yleensä tehokkaan efektin aikaansaamiseksi tarvitaan vähintään 3 tasoa: etuala, keskiala ja taka-ala. Näin saadaan esitettyä tasojen liikkeen nopeuden asteittainen muutos. Etualan tasojen nopeat liikkeet korostavat taka-alan hitaasti liikkuvien tasojen elementtien näennäistä skaalaa. Keskiala on tyypillisesti päätaso, joka liikkuu synkronoidusti kameran kanssa. (29).

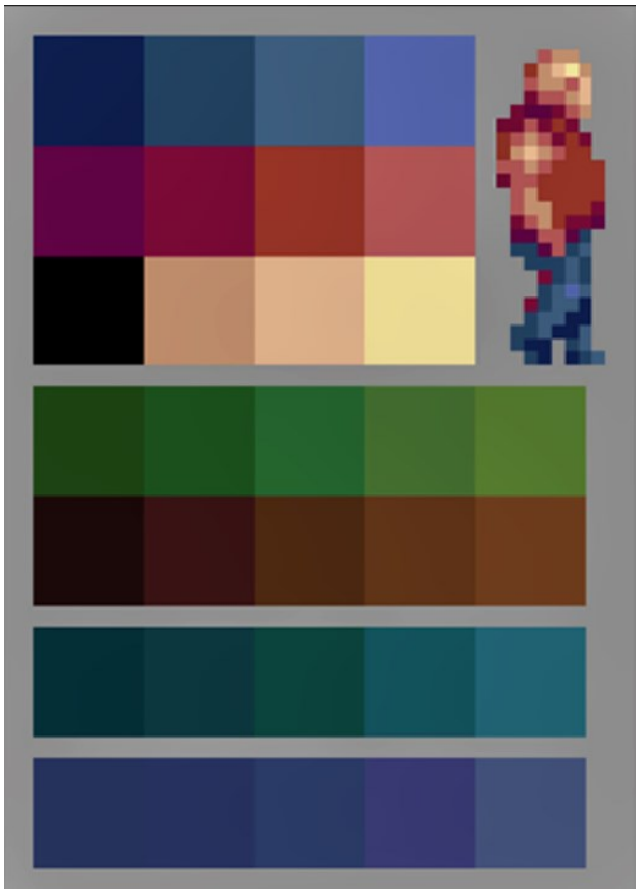
3.2.2 Kuinka vaikutelmaa voidaan visuaalisesti tehostaa

Parallaksi on tehokas syvyysvaikutelman luoja, sillä siinä syvyysvaikutelma perustuu elementtien liikkeeseen. Syvyysvaikutelmaa voidaan kuitenkin edesauttaa myös puhtaasti visuaalisin keinoin.

Valokuvauksessa syväterävyys on keskeisessä roolissa syvyysvaikutelmassa. Peliprojekteissa syväterävyyttä voidaan jäljitellä erilaisilla sumennuksilla. Sumennukset vaativat toimiakseen kuitenkin riittävän suuren resoluution, eli pikselitoteutukseen se ei suoraan kelpaa. Voimme kuitenkin hyödyntää tätä ajatuksen tasolla. Se, mihin pikselitoteutuksessa pystytään vaikuttamaan, on värien ja valöörin kontrasti sekä yksityiskohtien määrä. Kun etäisyys kameraan kasvaa, voimme sumentamisen sijaan siis vähentää edellä mainittuja kontrasteja sekä yksinkertaistaa taaempien tasojen grafiikkaa. Suuremmat kontrastit voidaan varata etualalle.

Kontrastin lisäksi myös värin sävyllä voidaan tehostaa syvyysvaikutelmaa imitoimalla ilmakehän vaikutusta siten, että taka-alan sävyt taittuvat näennäisen etäisyyden mukaan taivaan värin suun-

taan, eli yleensä sinertyvät (Kuva 11). Sininen väri perustuu siihen, kuinka valon sironta toimii, eli siniset aaltopituudet siroavat eniten. Mitä voimakkaampi ilmakehäefekti halutaan, sitä enemmän lähestytään taivaan sävyjä. (30).



KUVA 11. Pelaajan sekä taustalla olevien parallaksi tasojen värimaailma lähimmästä kauimmaiseen

4 TEKNISEN DEMON TOTEUTUS

4.1 Pelaajahahmo ja kamera

Tasohyppelypelin tärkein toiminnallinen kokonaisuus on pelattava hahmo sekä kamera, jonka kautta peliä havainnoidaan. Yleensä näiden elementtien toteutus on luonteva aloituspiste peliprojektille. Näistä toteutuksista aloitin myös tässä produktiossa.

Koin helpoimmaksi tavaksi toteuttaa pelaajahahmoa seuraavan kameran käyttäen hyväksi elementtien välistä hierarkiaa, sillä isäntäelementin koordinaatit periytyvät. Käytännössä tämä tarkoitti kameraobjektin asettamista hierarkiassa pelaajaobjektin kanssa samaan isäntäelementtiin. Aiemmassa luvussa totesimme, että lapsielementti perii isäntäelementin koordinaatit ja tämä pitää paikkaansa myös isäntäelementin liikkuessa. Tällöin lapsielementin omat koordinaatit toimivat offset-arvona. Tätä oppia hyväksikäyttäen produktiossa toteutettiin kameraskripti, jossa kameran sijainti kompensoi viiveellä siihen suuntaan, johon pelattava hahmo kullakin hetkellä katsoo. Yleensä tämä on toimivampi ratkaisu kuin täydellisen keskitetty kamera, sillä se tarjoaa pelaajalle paremman näkyvyyden kulkusuuntaan. Mikäli pelin kameralta kaipaa laajempaa heti valmista toiminnallisuutta, kuten elokuvamaisia kamera-ajoja, tarjoaa Unity myös kehittyneemmän työkalun nimeltä Cinemachine. Tarjolla on myös runsaasti kolmannen osapuolen tuottamia kameralaajennuksia. Tämän produktion tarpeisiin riitti kuitenkin yksinkertaisempi kamera, jota manipuloidaan skriptien kautta.

Jotta pelaajahahmo saatiin liikkeelle, liitettiin pelaajaobjektiin RigidBody2d-kontrolleri, sekä skripti, jolla kyseistä fysiikkaobjektia hallitaan saadun käyttäjäsyötteen mukaisesti. Skripti lisää määritettyjen näppäinten painalluksesta vauhtia fysiikkaobjektin X- ja Y-akseleille.

Koska kyse on tasohyppelypelistä, peliin haluttiin myös vertikaalista liikettä, eli myös hyppäämiselle tarvittiin toimiva toteutus. Mikäli pelaajahahmo on jo valmiiksi ilmassa, ei uudelleen hyppäämisen yleensä haluta olevan mahdollista. Hyppääminen toteutettiin implementoimalla tasohyppelypeleissä tyypillinen ”GroundCheck”, eli tarkistus siitä, onko pelaajahahmo maan tasalla. Tarkistus toteutettiin BoxCast-luotauksella, jolla tunnistetaan pelikentän laattojen törmäyttimet.

4.2 Muut toiminnalliset skriptit

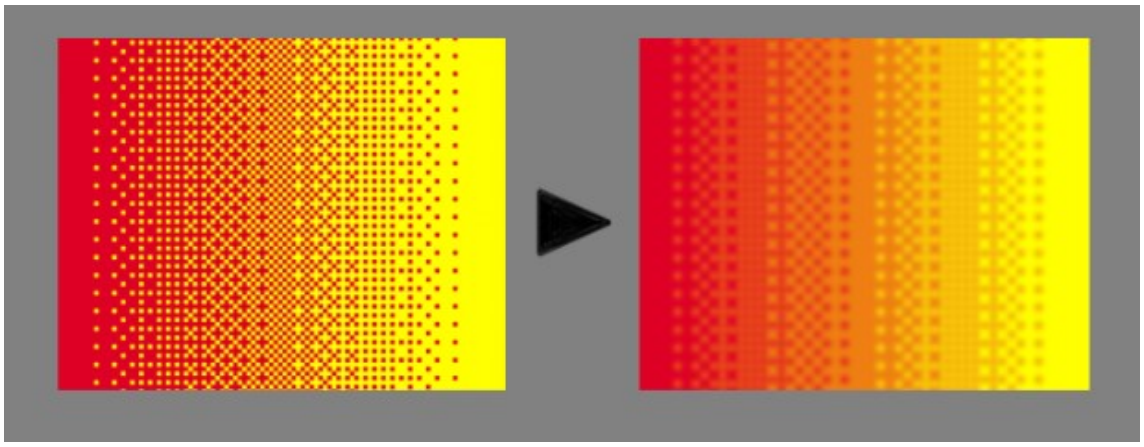
Produktion aikana toteutettiin myös muita pienempiä skriptitoteutuksia. Näistä mainitsemisen arvoisia ovat seuraavat:

- Skenen hallintaskripti, jolla dynaamisesti ladataan ja poistetaan skenejä ajonaikaisesti. Skene aktivoidaan tai deaktivoidaan pelaajan ohittaessa tietyn triggeripisteen. Triggereiden tunnistamaiseen käytetään tageja. Skripti toteutettiin SceneManager-luokan valmiita toimintoja hyödyntäen. (31).
- Liikkuvat tasot (eli Hissi), jossa pelaajaobjektin isäntäelementtiä muutetaan liikkuvalle tasolle astuessa. Tällöin pelin fysiikka toimii halutulla tavalla, eli liikkeen jatkuvuuden lakeja mukaillen.
- Yksinkertainen Vihollinen, joka pyrkii pelaajan luokse ja osaa kiivetä tiellään olevien esteiden yli boxcastiä hyödyntäen.

4.3 Graafiset valinnat

Pikseligrafiikka oli ennen pelikehityksessä teknisten rajoitteiden asettama vaatimus. Teknisiä rajoitteita grafiikalle asettivat muun muassa rajallinen tallennustila, resoluutio sekä käytössä oleva väriavaruus. Esimerkiksi 8-bittisen Nintendon eli NES-konsolin resoluutio on 256 × 240 ja käytettävissä oleva väriavaruus kattaa kaikkiaan 54 väriä. (32).

Rajallista väripalettia pystyi pikseligrafiikalla venyttämään käyttämällä erilaisia ”dithering”-menetelmiä, jotka toimivat samalla logiikalla kuin rasteripisteet printtimediassa. Pistetiheyttä säädellen saadaan emuloitua värisävyjä, joita ei todellisuudessa ole käytössä. Vanhoissa peleissä menetelmän väriliukumaa edesauttoi myös tuon aikaisten kuvaputkitelevisioiden epäterävä kuvanlaatu (kuva 12).



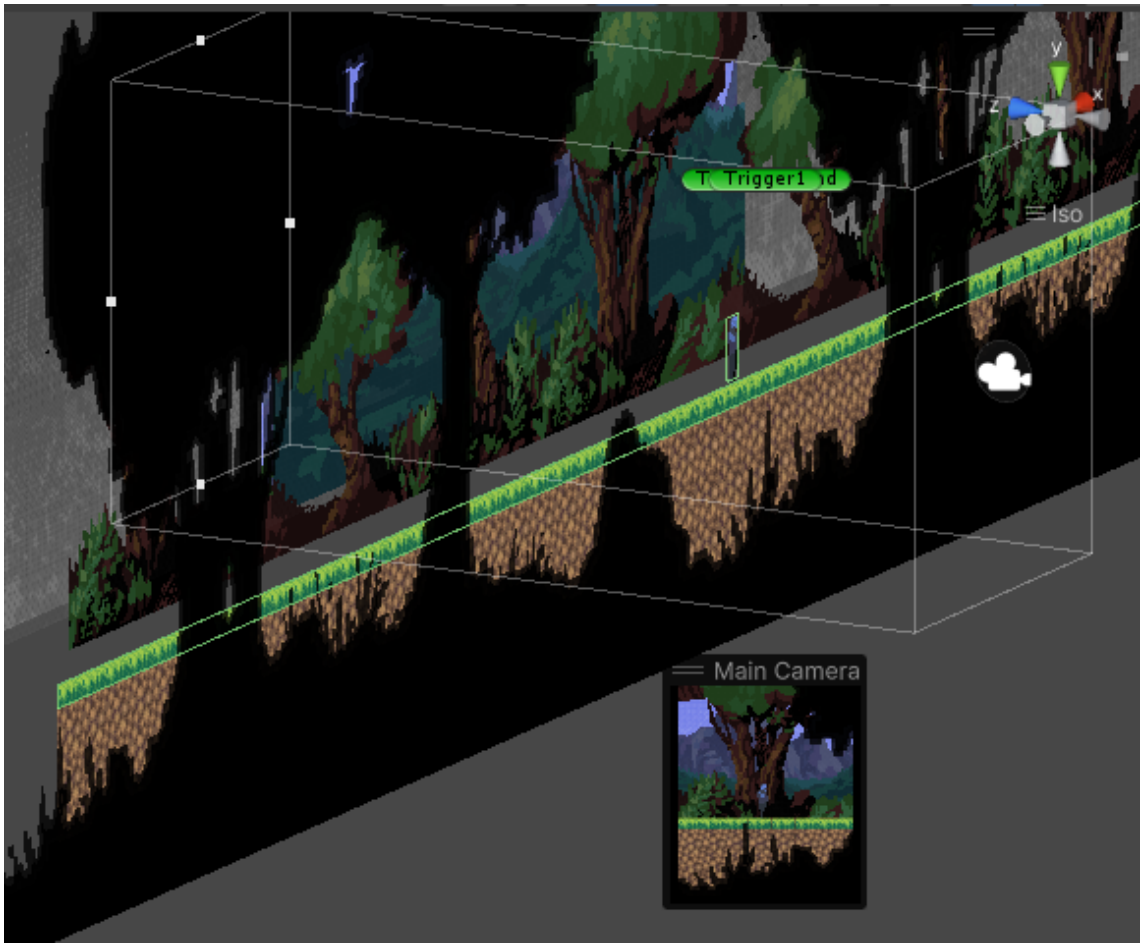
KUVA 12. Dithering-esimerkki. Vasemmalla liukuma punaisesta keltaiseen. Oikealla on sumennuksella havainnollistettuna mahdollinen lopputulos kuvaputkitelevision näytöllä.

Nykyään pikseligrafiikka ei ole pelikehityksessä tekniikan määräämä pakko, vaan tyylillinen ja resurssitekninen valinta. Tässä toteutuksessa (kuva 13) pikseligrafiikkaan päädyttiin rajatun resoluution mahdollistaman tehokkaan graafisten elementtien toteutuksen vuoksi, josta on hyötyä etenkin animoitujen resurssien luonnissa. Suurempi resoluutio moninkertaistaisi työmäärän animaatioruutua kohden. Pikseligrafiikan tehokkuutta lisää myös laattapohjainen toteutus, jossa pelikenttä rakentuu uudelleenkäytettävistä elementeistä. Samoin parallaksitaustat rakennettiin akseliltaan toistuviksi, jolloin työmäärä pysyi hallussa. Akseliltaan toistuessaan tausta ei välttämättä pääty halutusta kohtaa ja näkyviin voi jäädä puoliksi piirrettyjä elementtejä. Näitä korjattiin sprite-maskia hyödyntäen ja lopputulos oli varsin toimiva.

KUVA 13. Tekninen demo, työnimeltään ”Farewell” eli jäähyväiset

4.4 Parallaksi ja syvyysvaikutelma

Parallaksiefekti toteutettiin produktiossa viidessä tasoissa. Yleensä tehokkaan efektin aikaansaamiseksi tarvitaan vähintäänkin etuala, keskiala sekä taka-ala. Tässä toteutuksessa tehtiin etualan siluettitaso, keskiala eli päätaso, taustan kasvillisuustaso, puuraja, sekä viimeisenä horisontin vuoret (kuva 14).



KUVA 14. Parallaksitasot ja ortografinen kamera. Kuvan valkoinen kehikko havainnollistaa kameran piirtoaluetta, jonka lopputulos näkyy kuvankaappauksen ”Main Camera” ikkunassa

Tasot toteutettiin siten, että niitä pystyttiin toistamaan x-akselilla saumattomasti halutun pituuden aikaansaamiseksi. Koska kaksiulotteisessa toteutuksessa ei ole tarvetta Z-akselin datalle, käytettiin sitä tässä toteutuksessa Parallaksitasojen nopeuden kertoimena vaaka- ja pystyakseleille (kuva 15).

```
void Update()
{
    transform.position = new Vector3(startPos.x + (travel.x * zDistance), (startPos.y + travel.y * (zDistance/3)), zDistance);
}
```

KUVA 15. Z-akselin hyödyntäminen parallaksitason kertoimena

Näiden lisäksi toteutettiin alfakanavaa hyödyntäen läpinäkyvänä sääefektinä pilvet tehostamaan syvyyden tuntua. Pilviin liitettiin parallaksiskriptin lisäksi tekstuurin offset-arvoa inkrementoiva skripti (kuva 16), eli pilvet saatiin reagoimaan sekä kameran liikkeeseen että myös animoitua.

```
void Update()
{
    offset += speed * Time.deltaTime;
    material.mainTextureOffset = new Vector2(offset, offset);
}
```

KUVA 16. Pilvien tekstuurin offset-arvoa päivittävä yksinkertainen mutta visuaalisesti toimiva toteutus

4.5 Lopputuloksen arviointi

Produktion lopputuloksena syntyi toimiva mutta skoopiltaan suppea tekninen demo. Toteutus oli erittäin opettavainen katsaus pelikehitykseen. Painopiste oli graafisessa puolessa johtuen erityistarkastelun kohteesta, eli varsinainen kooditoteutus jäi melko yksinkertaiselle tasolle. Näen toteutuksen olevan kuitenkin toimiva pohja myös skriptien osalta.

Pikseligrafiikalla on monia etuja pelituotannossa, mutta erityisesti siinä kiehtoo sen ekonomisuus. Pikseligrafikkaa voi tuottaa nopeasti ja pienillä henkilöresursseilla. Tämä mahdollistaa myös yhden miehen studion kuten tässä projektissa. Sekä koodi että grafiikka syntyvät itsenäisen työn tuloksena. Graafisissa elementeissä piilee tietty variaatio vähenevän tuoton laista. Joka kerralla, kun resoluutio tuplataan, työstä saatava esteettinen hyöty vähenee suhteessa työmäärään.

5 POHDINTA

Opinnäytetyössä tutkittiin kaksiulotteisen tasohyppelypelin kehittämistä Unity-pelimoottorilla. Opinnäytetyön produktiona toteutettiin tekninen demo kaksiulotteiselle tasohyppelypelille, jota on mahdollista käyttää pohjana idean jatkojalostuksessa. Produktion puitteissa valmistui pelaajahahmon liikkumisen skriptitoteutus, sekä kameraskripti. Peliin toteutettiin suurempana kokonaisuutena parallaksiskripti taustoille sekä tätä toteutusta mukaillen myös vastaava skripti sääefekteille. Lisäksi luotiin pohjatoteutus skenejen dynaamiselle vaihdolle.

Produktion aikana nostettiin esiin useampaan kertaan kysymyksiä pikselipohjaisten estetiikan arvostuksesta kuluttajien keskuudessa, eli toisin sanoen pohdittiin, löytyykö pikseligrafiikalla toteutetulle pelille ostajia. Pohdinnassa päädyin nopeasti lopputulokseen, ettei se ole ainakaan kaupallisen menestyksen este, mikäli peli on mekaniikaltaan riittävän kiinnostava. Tämän väittämän perustelemiseksi ei tarvitse lähteä kotimaata kauemmaksi, eikä edes ajassa tarvitse matkustaa takaisin Habbo Hotellin kulta-aikaan. Yksi kotimaassa vähälle julkisuudelle jääneistä suomalaisista menestystarinoista on Nolla Games, jonka taustalla vaikuttaa ryhmä Suomen pelialan ammattilaisia, joiden näppäimistöiltä on syntynyt koko joukko tunnettuja pelejä. Pelkästään rahassa mitattuna menestynein peleistä lienee Noita, joka on haastava mutta koukuttava ”roguelite”. Harvassa suomalaisessa yrityksessä löytyy prosentuaalisesti yhtä monta taustavaikuttajaa, joiden nimet palautuvat verokoneesta, eli ainakaan heidän menestymistään ei pienen resoluution estetiikka ole estänyt.

Aloitin tämän opinnäytetyön produktion (kuva 17) tuottamisen alkuvuodesta 2021, eli tein valinnan produktiossa käytettävästä pelimoottorista jo tuossa vaiheessa. Sittemmin Unityn suunnitelmat yksipuoleisesta ja takautuvasta muutoksesta lisenssiehtoihin syksyllä 2023 aiheuttivat kehittäjäyhteisössä ja pelialan yrityksissä närää ja asettivat Unityn ammattimaisena alustana kyseenalaiseen valoon. Mikäli projektia jollain tasolla jatkan, pitää myös kehitystyössä käytettävää pelimoottoria harkita uudelleen.

KUVA 17. Farewell

LÄHTEET

1. Unity Technologies 2024. Unity's interface. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/UsingTheEditor.html
2. Unity Technologies 2024. Scenes. Hakupäivä 15.5.2024. https://docs.unity3d.com/2022.3/Documentation/Manual/CreatingScenes.html
3. Unity Technologies 2024. Work with multiple scenes in Unity. Hakupäivä 15.5.2024. https://docs.unity3d.com/2022.3/Documentation/Manual/MultiSceneEditing.html
4. Unity Technologies 2023. GameObjects. Hakupäivä 15.1.2024. https://docs.unity3d.com/2022.2/Documentation/Manual/GameObjects.html
5. Unity Technologies 2024. The Hierarchy window. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/Hierarchy.html
6. Unity Technologies 2024. Physics 2D Reference. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/Physics2DReference.html
7. Unity Technologies 2024. Introduction to Rigidbody 2D. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-Rigidbody2D.html
8. Unity Technologies 2024. Collider 2D. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/Collider2D.html
9. Unity Technologies 2024. PhysicsMaterial2D. Hakupäivä 16.5.2024. https://docs.unity3d.com/ScriptReference/PhysicsMaterial2D.html
10. Unity Technologies 2024. Tilemap Collider 2D. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-TilemapCollider2D.html
11. Unity Technologies 2024. Composite Collider 2D. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-CompositeCollider2D.html
12. Unity Technologies 2024. Physics2D.Raycast. Hakupäivä 16.5.2024. https://docs.unity3d.com/ScriptReference/Physics2D.Raycast.html
13. Unity Technologies 2024. Physics2D.BoxCast. Hakupäivä 16.5.2024. https://docs.unity3d.com/ScriptReference/Physics2D.BoxCast.html
14. Microsoft 2024. Access Modifiers (C# Programming Guide). Hakupäivä 16.5.2024. https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers
15. Unity Technologies 2023. MonoBehaviour. Hakupäivä 12.4.2024. https://docs.unity3d.com/2022.2/Documentation/ScriptReference/MonoBehaviour.html

16. Unity Technologies 2024. Time. Hakupäivä 16.5.2024. https://docs.unity3d.com/ScriptReference/Time.html
17. Unity Technologies 2024. Textures. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/Textures.html
18. Unity Technologies 2024. Sprite (2D and UI) Import Settings reference. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/texture-type-sprite.html
19. Microsoft 2021.Nearest-point sampling. Hakupäivä 16.5.2024. https://learn.microsoft.com/en-us/windows/uwp/graphics-concepts/nearest-point-sampling
20. Gambetta, Gabriel 2021. Computer Graphics from Scratch. O'Reilly. Hakupäivä 16.5.2024. https://learning.oreilly.com/library/view/computer-graphics-from/9781098128968/xhtml/ch14.xhtml#ch14lev2.
21. Unity Technologies 2024. Work with sprites. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/Sprites.html
22. Unity Technologies 2024. Sprite Atlas. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/sprite-atlas.html
23. Unity Technologies 2024. Tilemaps. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/Tilemap.html
24. Unity Technologies 2024. Grid component reference. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-Grid.html
25. Unity Technologies 2024. Sprite Masks. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-SpriteMask.html
26. Unity Technologies 2024. Animator Controller. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-AnimatorController.html
27. Unity Technologies 2024. Blend Trees. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-BlendTree.html
28. Unity Technologies 2024. Camera component. Hakupäivä 16.5.2024. https://docs.unity3d.com/Manual/class-Camera.html
29. Younis, Adam 2021.The Perfect Pixel Art Parallax Tutorial [and Unity script!]. Hakupäivä 16.5.2024. https://www.youtube.com/watch?v=tMXgLBwtsvI
30. Gurney, James. 2010. Color and Light. James Gurney. Andrews McMeel Publishing
31. Unity Technologies 2024. SceneManager. Hakupäivä 16.5.2024. https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html

32. Loguidice, Bill & Barton, Matt 2014. Vintage Game Consoles. O'Reilly. Hakupäivä 16.5.2024. https://learning.oreilly.com/library/view/vintage-game-consoles/9780415856003/xhtml/20_Chapter09.xhtml#ch9.