Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Joona Rytkönen

Tasohyppelypeli

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

16.11.2020

Tekijä Otsikko Sivumäärä Aika	Joona Rytkönen Tasohyppelypeli 45 sivua 16.11.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaaja	Lehtori Miikka Mäki-Uuro

Opinnäytetyön tarkoituksena oli kehittää tasohyppelypeli Unity-pelimoottoria apuna käyttäen ja oppia yleisesti pelinkehitysprosessista. Työssä verrattiin indie-pelien eroavaisuuksia AAA-peleihin ja analysoitiin, millä eri tavoilla ne eroavat toisistaan.

Tasohyppelypelit ovat videopelien lajityyppi, joka keskittyy pääasiassa pelaajan hallitsemaan pelihahmoon ja sen kykyyn liikkua ja hyppiä esteitä vältellen. Työssä perehdyttiin siihen, miten pelin mekaniikat ja eri ominaisuudet toimivat ja sovellettiin niitä insinöörityöprojektiin. Työssä keskityttiin pääasiassa kenttäsuunnitteluun ja ohjelmointiin.

Työssä perehdyttiin myös nykypäivän pelinkehitysprosessiin. Tavoitteena oli ottaa selvää, mitä resursseja se vaatii ja mistä eri vaiheista pelinkehitysprosessi koostuu. Nykypäivän pelinkehitysprosessi koostuu esituotanto-, tuotanto- ja jälkituotantovaiheista. AAA-pelien kehitysprosessi työllistää satoja ihmisiä, ja kehitysprosessi kestää useita vuosia. AAA-peleissä on usein iso kustantaja takana, joka määrittää tarkan aikataulun kehitysprosessille. Indie-pelejä kehittävät joko yksityishenkilöt tai pienet tiimit, joilla on harvoin taustalla taloudellinen tuki kustantajalta.

Projektista syntyi toimiva peliprototyyppi, jonka voi pelata alusta loppuun ilman peliä rikkovia ongelmia. Projektin tarkoituksena oli kehittää pelinkehitystaitoja ja auttaa työnhaussa tulevaisuudessa.

Avainsanat	tasohyppely, Unity, pelinkehitys, pelimoottori

Author Title Number of Pages Date	Joona Rytkönen Platformer Game Development 45 pages 16 November 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Game Applications
Instructors	Miikka Mäki-Uuro, Senior Lecturer

The goal of this thesis was to develop a prototype of a puzzle-platformer-type game using Unity game engine and to learn about the game development process. Another aim was comparing indie-games to triple-A games and how they differ from each other.

Platformer-games are genre of video games that mainly focus on the player-controlled game character and its ability to run and jump to avoid obstacles. I will be explaining the mechanics and the thought process of how I implemented these characteristics to my project. This thesis mainly focuses on level design and programming.

This thesis also looked at today's game development process. The goal was to find out what kind of resources it requires to make a video game and what the different game development stages it consists of. The current game development process includes a pre-production-, production- and a post-production phase. The development of triple-A games employs hundreds of people, and the development process itself takes several years. Triple-A games often have a big publisher behind them that sets a precise schedule for the development team. Indie games, on the other hand, are developed by either individuals or small teams that rarely have the financial support of a publisher.

As a result of this final year project a working prototype which can be played from start to finish without any game breaking problems was created. The purpose of the project was to further develop game development skills and to help with work employment in the future.

Keywords	Platformer, Unity, Game Development, Game Engine

Sisällys

1 Johdanto

Opinnäytetyön tarkoituksena oli kehittää tasohyppelypeli Unity-pelimoottorilla PC-alustalle. Tarkoitus oli ottaa selvää, kuinka Unityä voi käyttää 2.5D-tasohyppelypelin toteutuksessa ja perehtyä Unityyn syvällisemmin. Samalla määriteltiin indie- ja AAA-pelien eroja ja tutustuttiin ”ison budjetin” eli AAA-pelien pelinkehitysprosessiin.

Unity Technologiesin (1) toimitusjohtajan mukaan puolet maailman peleistä on tehty Unityllä. Se on erityisen suosittu pelimoottori mobiilipelikehittäjien keskuudessa. Unityä käytetään yhä enemmän 3D-suunnitteluun ja simulaatioihin muillakin toimialoilla, kuten elokuva-, auto- ja arkkitehtuuriteollisuudessa. Sitä käytetään luomaan 60 % kaikista lisätyn ja virtuaalitodellisuuden kokemuksista. (2.) Unity valittiin tähän projektiin, koska tekijällä oli kokemusta siitä opinnoissa tehdyistä projekteista.

Tämän projektin toteuttamiselle oli lukuisia syitä. Suurimpana oli varmasti se, että insinöörityön tekijä on pelannut videopelejä yli kaksikymmentä vuotta ja peliala on aina kiinnostanut suuresti. Ideana oli tehdä projekti, josta voisi olla hyötyä tulevaisuudessa työnhakuprosessissa. Tarkoituksena oli myös tutustua pelisuunnitteluprosessiin ja oppia lisää ohjelmoinnista.

Luvussa 2 tutustutaan indie- ja AAA-peleihin ja verrataan niiden eroavaisuuksia. Itse AAA-pelinkehitysprosessiin tutustutaan luvussa 3 ja käydään läpi sen eri vaiheet ja se, mitä resursseja prosessi vaatii. Sen jälkeen luvussa 4 perehdytään Unityyn ja käydään läpi, mikä on pelimoottori ja mihin sitä käytetään. Luvuissa 5 ja 6 käydään läpi peliprojektin kehitys. Ensin suunnitteluvaihe luvussa 5 ja sen jälkeen projektin toteutus luvussa 6.

2 Indie- ja AAA-pelien määrittely

Pelien kehittäminen vaatii paljon työtä ja rahaa. Suurten pelien budjetti on usein kymmeniä miljoonia euroja, ja niiden parissa työskentelee kymmenistä ihmisistä satoihin ihmisiin. Pelin voi kuitenkin kehittää vain muutaman ihmisen voimin, tai jopa yksinkin (3). Useimmat paljon resursseja vievät pelit vaativat tarkan suunnitelman, jossa määritellään tietyt vaiheet, jotka on suoritettava pelin tuottamista varten. Pelinkehitysprosessissa ei kuitenkaan ole mitään tiettyä tapaa toimia, vaan jokaisella tiimillä on omanlaisensa prosessi, jota ne noudattavat. Pelinkehityksessä käytetään yleensä kahta eri termiä kehitysprojekteista, ”indie” ja ”AAA”. (4.)

Pelin käsite on laaja ja edellyttää tarkempaa määrittelyä. Mitchellin (5, s. 1) mielestä pelin perusteet voi määritellä siten, että johonkin aktiviteettiin liitetään haaste-elementti ja sille laaditaan joukko sääntöjä. Gregory (6, s. 8–9) siteeraa teoksessaan Raph Kosteria, joka on määritellyt teoksessaan Theory of Fun for Game Design pelin käsitteen seuraavasti: ”Peli on interaktiivinen kokemus, joka tarjoaa pelaajalle yhä haastavampia kaavasekvenssejä, joista pelaaja oppii ja lopulta hallitsee”. Kuvailtaessa PC- tai konsolipelejä termi yleensä tuo mieleen kolmi- tai kaksiulotteisia virtuaalimaailmoja, jossa pelaaja kontrolloi esimerkiksi ihmistä tai ajoneuvoa.

Nykyään markkinoilla on valtava määrä pelejä, mistä valita. Tästä huolimatta suurin osa niistä voidaan jakaa kahteen ryhmään niiden pelikehityksen perusteella. Nämä tyylit ovat hyvin erilaisia. Ne vaihtelevat hinnan, graafisen suunnittelun ja tiedostokoon mukaan.

Indie-pelit

Indie-pelejä kehittävät joko yksityishenkilöt tai pienet tiimit, joilla on harvoin taustalla taloudellinen tuki kustantajalta. Tiimien koko vaihtelee muutamasta henkilöstä muutamaan kymmeneen henkilöön. Monet indie-kehittäjät luottavat joukkorahoitukseen, esimerkiksi suositun yhdysvaltalaisen joukkorahoituspalvelun, Kickstarterin (7). kautta. Indie-kehittäjillä ei ole valtavaa budjettia käytössään, ja täten pelit ovat yleensä kooltaan pienempiä ja lyhyempiä. Indie-pelien menestyksen takana on yleensä innovaatio, kuten tyylitelty taidesuunnittelu tai uniikki pelimekaniikka. (8.)

Indie-pelit ovat helpommin lähestyttäviä suurimmalle osalle pelaajista kuin ison budjetin AAA-pelit. Ensinnäkin ne ovat paljon halvempia kuin AAA-pelit, sillä ne maksavat harvoin yli 20 euroa. Pelattavuus niissä on myös suhteellisen helppoa ja yksinkertaista. Niiden yksinkertaistetun luonteen ansiosta monet pelaajat voivat nauttia niistä, pelitaidosta ja kokemuksesta riippumatta. PC-pelaajat voivat pelata niitä, vaikka heillä ei olisikaan markkinoiden uusimpia prosessoreita tai näytönohjaimia, sillä ne yleensä vaativat vähemmän tehoa tietokoneelta. (8.)

Playdead on suosittu indie-kehittäjä, joka tunnetaan tasohyppelypeleistään, joissa on uniikki taidetyyli. Sen ensimmäinen peli Limbo (kuva 1) julkaistiin vuonna 2010. Studiolla oli 8 henkilön kehitystiimi, joka laajeni enimmillään 16 henkilöön eri kehitysvaiheissa freelancereiden voimin. Pelin kehitys kesti kaksi ja puoli vuotta, ja sitä pidetään yhtenä parhaimmista indie-peleistä. Limbo toimi isona inspiraationa insinöörityön pelin ideoinnissa. (9.)

Kuva 1. Playdead-pelistudion kehittämä indie-peli Limbo (10).

AAA-pelit

AAA-pelit ovat täysin vastakohtia indie-peleille. Niitä kehittävät suuret studiot, joissa työskentelee satoja tai jopa tuhansia ihmisiä. Lisäksi hankkeita tukee julkaisija, joka toimittaa kehitystiimille riittävän budjetin. Tunnettuja isoja julkaisijoita ovat mm. Activision Blizzard, Ubisoft ja Electronic Arts. Budjetin ja kehitystiimien ison koon vuoksi AAA-pelit ovat yleensä pitkiä, suuria ja niissä on yksityiskohtainen ja realistinen pelimaailma. Termi AAA itsessään on tarkoitettu luomaan odotuksia siitä, että nämä pelit ovat erittäin korkealaatuisia. (8.)

Näihin peleihin menevän paljon suuremman rahallisen sijoituksen seurauksena pelit maksavat yleensä enemmän kuin indie-pelit. AAA-pelien hinnat vaihtelevat tyypillisesti 40 eurosta 70 euroon. Niiden pelattavuus on myös yleensä paljon monimutkaisempaa ja laajempaa kuin indie-peleissä. AAA-peleissä on yleensä huippuluokan grafiikka, joten PC-pelaajat tarvitsevat riittävän tehokkaan laitteiston pelatakseen niitä. (8.)

Jeff Grubbin artikkelin (3) mukaan Ubisoftilla, joka on yksi maailman isoimpiin pelitaloihin kuuluvista yrityksistä, on käytössään satojen henkilöiden tiimi kehittäessään peliä Playstation 4 -konsolille. Kuvassa 2 nähdään kuvakaappaus Ubisoftin kehittämästä Assassin’s Creed Odysseystä.

Kuva 2. Ubisoftin kehittämä Assassin's Creed Odyssey (11).

Ubisoftin toimitusjohtajan Pauline Jacqueyn (3) mukaan Ubisoft on erikoistunut kehittämään avoimen pelimaailman toimintapelejä, jotka vaativat paljon resursseja ja jotka työllistävät jopa 400–600 henkilöä.

3 AAA-pelin kehitysprosessi

Ison budjetin pelin eli AAA-pelin kehitysprosessi koostuu monista vaiheista. Ensin on esituotantovaihe, joka tapahtuu ennen kehityksen alkamista. Sitten tulee tuotantovaihe, jossa varsinainen peli kehitetään. Lopuksi päästään jälkituotantovaiheeseen, joka tulee pelin kehityksen jälkeen. Siinä tehdään tarvittavat korjaukset peliin.

3.1 Kehitysprosessin esituotanto

Kehitysprosessin esituotanto on AAA-pelinkehitysprosessin suunnittelu- ja valmisteluvaihe ennen varsinaisen kehityksen alkamista.

Suunnittelu

Suunnitteluvaihe tapahtuu ennen varsinaisen kehityksen alkamista. Pohjimmiltaan suunnitteluvaihe määrittelee, mistä pelissä on kyse, miksi se pitäisi tehdä ja mitä resursseja sen tekeminen vaatii. Pickellin (12) mukaan pelin ideointi on yksi vaikeimmista pelinkehityksen vaiheista, sillä se toimii koko pelin selkärankana. Se asettaa standardin jokaiselle pelinkehityksessä mukana olevalle työntekijälle ja antaa kustantajille kuvan siitä, mitä peliltä odottaa:

- Mistä pelissä on kyse?
- Kenelle peli on suunnattu?
- Onko sille kysyntää?
- Kauan sen kehittäminen kestää?
- Kuinka paljon resursseja se vie?

- Mikä on budjettiarvio?

Suunnitteluvaihe voi kestää viikosta jopa vuoteen, riippuen projektin koosta, käytettävissä olevista resursseista ja budjetista. Siihen voi kulua 20 % projektin kokonaisajasta. (13.)

Konseptitodistus

Konseptitodistus edustaa todisteita siitä, että projekti tai tuote on toteuttamiskelpoinen ja riittävän kelvollinen perustelemaan sen tukemiseen ja kehittämiseen tarvittavat varat. Pelisuunnittelussa kerätään kaikki kehitetyt ideat ja arvioidaan, kuinka todennäköisesti pelistudio voi ne toteuttaa. Tästä herää lisäkysymyksiä, esim.

- Mitkä ovat arvioidut kustannukset?
- Onko tiimillä teknisiä valmiuksia pelin kehittämiseen?
- Tarvitaanko uusi pelimoottori?
- Kuinka suuri tiimi tarvitaan projektin kehittämiseen?
- Tarvitaanko ulkopuolista tukea?
- Kuinka peli kaupallistetaan?

Studiolle, joka kehittää peliä kustantajan alaisuudessa, konseptin oikeaksi todistaminen on välttämätöntä, ennen kuin pelin kehitystä voidaan jatkaa. (13.) Tämä johtuu siitä, että kustantajan on hyväksyttävä kehityksen aikataulu, budjetti ja markkinointipuoli. Indie-studioille, joilla ei ole julkaisijoiden valvontaa, tässä vaiheessa on hieman enemmän joustavuutta. Itsenäisen julkaisemisen ongelma on kehitys- ja markkinointibudjetin luominen. (14.)

Game Design Document

Pelinkehitystä varten luodaan pelisuunnitteludokumentti eli Game Design Document. Siihen merkitään pelin tekniset tiedot, joissa on enemmän yksityiskohtia kuin konseptissa. Se on niin sanottu ”elävä asiakirja”, johon viitataan ja jota muokataan tuotannon aikana. (4.) GDD pitää kehitysprosessin järjestäytyneenä ja se auttaa tunnistamaan mahdolliset riskit (13).

Pelin prototyyppi

Videopeliprototyyppi on testi, joka tarkistaa toimivuuden, käyttökokemuksen, pelattavuuden, mekaniikan ja taiteen suunnan. Pelin prototyypin tekeminen tapahtuu esituotantovaiheessa, jotta voidaan testata, toimiiko pelin idea käytännössä, ja onko sen kehitystä kannattavaa jatkaa. Monet ideat eivät pääse tähän vaiheeseen. Kehitystiimi aloittaa usein paperisuunnitelmilla testatakseen teorioita ja selvittääkseen pelin vivahteet nopeasti, helposti ja kustannustehokkaasti. (13.)

Vaikka käsitteet, ideat ja teoriat ovat tärkeitä asioita, niitä voi kehittää vain tiettyyn pisteeseen asti paperilla. Tavoitteena on saada prototyyppi toimimaan mahdollisimman nopeasti, jotta voidaan testata, toimivatko peliin suunnitellut ideat käytännössä. (13.) Prototyyppi voi olla suurin yksittäinen vaikuttaja siinä, jatkuuko projekti lainkaan. Jos kustantajat eivät näe pelin visiota heti prototyypistä, sitä ei todennäköisesti tulla rahoittamaan. (15.)

3.2 Tuotanto

Tuotanto on kehityksen pisin vaihe. Yleensä kestää yhdestä vuodesta neljään vuoteen, ennen kuin peli alkaa muotoutua. Suurin osa pelin kehittämiseen käytetystä ajasta, vaivasta ja resursseista kuluu tuotantovaiheessa. Se on yksi kehityksen haastavimmista vaiheista. Ei ole harvinaista, että kokonaisia pelisegmenttejä, joihin on mennyt kuukausia kehittää, jää käyttämättömiksi niiden valmistuttua. (12.)

Projektipäällikkö tai pelituottaja on vastuussa hyvästä koordinaatiosta tiimin jäsenten välillä. Hänen on varmistettava projektin sujuvuus, ennakoitava ja ratkaistava riskitilanteita. Peliohjelmoijat kirjoittavat tuhansia riviä koodia saadakseen jokaisen pelisisällön toimimaan. Suunnittelijat luovat pelihahmot näyttämään ja liikkumaan täsmälleen, niin kuin pelin tarinassa niiden pitäisikin. Äänisuunnittelijat varmistavat, että pelin äänet kuulostavat aidoilta. Esimerkiksi, kun hahmo kävelee nurmikolla, juoksee sementillä tai puhuu, äänten täytyy kuulostaa autenttisilta. Tasosuunnittelijat luovat ympäristöjä tai tasoja, jotka sopivat tarinan teemaan. (16.)

Tuotanto päättyy tyypillisesti ennalta määrättyyn päivään. Tällä varmistetaan, että peli on julkaistu ennen taloudellisesti merkittävää tapahtumaa. Muutaman studiot saattavat olla halukkaita pitämään tuotannon käynnissä, kunnes saavutetaan tietty laatutaso, mutta tämä ei valitettavasti ole normi AAA-peleissä, joissa on isoja yrityksiä tuottajina. (17.)

Testaus

Pelitestaus on olennainen osa pelin kehitysprosessia, on sitten kyseessä pieni tai iso projekti. Ne ovat välttämättömiä pelisuunnittelulle, koska jokaisen pelinkehittäjän tai kehitystiimin on kuultava sellaisten ihmisten näkökulma pelistä, jotka eivät ole mukana varsinaisessa kehitysprojektissa. (18.)

Pelitestaajat joutuvat ottamaan huomioon seuraavanlaisia kysymyksiä:

- Onko pelissä ongelmallisia alueita tai tasoja?
- Näkyvätkö kaikki peliobjektit oikein ruudulla?
- Voiko seinien tai muiden kiinteiden objektien läpi kävellä?
- Voiko pelissä jäädä jumiin johonkin kohtaan?
- Toimivatko äänet ja animaatiot kunnolla?
- Kaatuuko peli?

Jokainen pelin ominaisuus ja mekaniikka on testattava laadun takaamiseksi. Peli, jota ei ole testattu perusteellisesti, ei ole valmis edes alfajulkaisuun. (12.)

3.3 Jälkituotanto

Markkinointi

Videopeliä markkinoidaan koko sen jälkituotannon aikana, ja markkinointi jatkuu vielä jonkin aikaa pelin julkaisun jälkeen. Suuret tuotantotalot yleensä aloittavat pelin markkinoinnin tuotannon loppupäässä saadakseen näkyvyyttä ja herättääkseen fanien mielenkiinnon. (13.)

Ylläpito

Kun tuotanto on valmis ja peli on myyntivalmis, sen kehittämisprosessi jatkuu. Jotkut tiimin jäsenet siirretään ylläpitoon, ja toiset taas siirretään esimerkiksi tekemään jatko-osaa peliin tai ihan uuteen projektiin. (13.) Ylläpitoon kuuluu pelin jälkeisten virheiden korjaaminen ja kaiken pelin jälkeisen sisällön käsittely. Tuotannon jälkeinen työ ilmenee pelin sisäisinä korjaustiedostoina ja ladattavana sisältönä. (4.)

4 Yleiskatsaus Unity-pelimoottoriin

Tässä luvussa määritellään ensin yleisesti, mikä on pelimoottori, ja käydään läpi modernien pelimoottorien hyödyt. Tämän jälkeen tutustutaan Unityn käyttöliittymään, josta saa hyvän yleiskuvan siitä, miten Unity-editorissa työskennellään.

4.1 Pelimoottorin määritelmä

Termi ”pelimoottori” syntyi 1990-luvun puolivälissä, ja se liittyi FPS-peleihin, kuten erittäin suosittuun Doom-peliin. Pelimoottorien tehtävänä on muun muassa grafiikan renderöinti, törmäyksen havainnollistaminen ja peliäänien tuottaminen. Doomin arkkitehtuurissa onnistuttiin erittelemään nämä ohjelmistokomponentit pelin resursseista, maailmoista ja säännöistä, jotka muodostivat pelaajan pelikokemuksen. Tämän seurauksena pelinkehittäjät hankkivat muiden pelien lisenssejä ja alkoivat muokata niitä uusiksi tuotteiksi luomalla uutta taidetta, kuten aseita, pelihahmoja ja uusia pelimaailmoja. Pelimoottorin ohjelmistoon ei tarvinnut tehdä kuin minimaalisia muutoksia, sillä kehittäjät pystyivät käyttämään siihen rakennettuja ohjelmistokomponentteja hyväkseen. (6, s. 8–9.)

4.2 Unity pähkinänkuoressa

Unity on tehokas integroitu pelimoottori ja editori, jonka avulla voi luoda objekteja, tuoda ulkoisia resursseja pelinkehittäjän käyttöön ja yhdistää ne toimivaksi kokonaisuudeksi koodin avulla. Unityssä on sisäänrakennettu Monodevelop-ohjelmointialusta Linuxille,

macOS:lle ja Windowsille, ja sen avulla voidaan luoda ohjelmakoodia peliä varten. Nykyään kuitenkin oletusohjelmointialusta Unityssä on Microsoft Visual Studio, joka voi asentaa Unityn mukana. Unityn avulla saa myös käyttöönsä verkko-ominaisuudet moninpelikehitystä varten ja mahdollisuuden rakentaa pelin monelle alustalle. Unity tukee seuraavia alustoja:

- PC, Mac ja Linux
- tvOS
- PS4
- iOS
- Xbox One
- Android
- WebGL.

Unityn käyttöliittymä perustuu ns. vedä ja pudota -periaatteeseen, jossa käyttäjä voi muun muassa yhdistää skriptejä tai määrittää objekteille referenssejä raahaamalla ne hiiren osoittimella oikeisiin paikkoihin. (19, s. 3.)

Unity hyödyntää graafisia API-ohjelmointirajapintoja, kuten Direct3D, OpenGL, Vulkan ja Metal, ja sitä on käytetty kehittämään tuhansia PC-, mobiili- ja konsolipelejä vuoden 2005 ensijulkaisun jälkeen. (20, s. 10.)

Modernien pelimoottoreiden hyödyt

Pelimoottorien avulla pystytään vähentämään videopelien kehityksen kustannuksia, monimutkaisuutta ja pelin tuottamiseen vaadittavaa aikaa. Ne tarjoavat valtavia etuja tehokkuutta ajatellen, vähentämällä pelien kehittämiseen tarvittavaa tietämystä. Modernit pelimoottorit erottavat erilaiset toiminnot sisäisesti. Esimerkiksi kehittäjän tekemä pelikoodi, joka kuvaa pelaajan liikkumista 3D-ulottuvuudessa tai pelissä kuuluvien peliäänten logiikkaa, pidetään erillään koodista, joka purkaa äänitiedoston ja lataa sen muistiin. (20, s. 8–9.)

Modernit pelimoottorit sisältävät osan tai kaikki seuraavanlaisista ominaisuuksista:

- grafiikan renderöintimoottori, joka tukee 2D- tai 3D-grafiikkaa
- fysiikkamoottori, joka tukee törmäyksien havaitsemista
- äänimoottori äänien ja musiikkitiedostojen lataamiseen ja toistamiseen
- ohjelmointikirjastoja pelikoodin implementoinnin helpottamiseksi
- sisäänrakennettu animointityökalu
- muistinhallinta
- tekoäly polunetsintää ja NPC-hahmoja varten.

Nämä ominaisuudet nopeuttavat huomattavasti kehitysprosessia.

4.3 Käyttöliittymä

Unityn käyttöliittymä koostuu useista näkymistä (kuva 3).

Kuva 3. Unity-editorin käyttöliittymä. 1. Työkalupalkki. 2. Hierarkianäkymä. 3. Scene-näkymä. 4. Inspector-näkymä. 5. Projektinäkymä (Unity editor).

Työkalupalkki

Työkalupalkin avulla pääsee käsiksi oleellisimpiin työominaisuuksiin. Sen alapuolella on navigointityökalurivi, jossa on kolme painiketta. Play-painikkeesta pääsee ”Play Modeen”, jossa voi testata peliä ja pelin voi vastaavasti pysäyttää stop-painikkeella. Tämän avulla peliä voi testata nopeasti ja toistuvasti ja samalla varmistaa, että peli toimii ja virheilmoituksia ei tule. (21, s. 104.)

Scene-näkymä

Jared Halpernin mukaan (19) skenejä eli näkymiä voidaan pitää Unity-projektien perustana. Unity-editoria käytettäessä suurin osa ajasta kuluu tämän näkymän parissa. Kaikki pelissä tapahtuva tapahtuu kohtauksessa. Peli voi koostua yhdestä tai useammasta kohtauksesta, ja jokaisen kohtauksen peliobjektit, eli ympäristöt, hahmot, valot, kamera ja kaikki, mistä peli koostuu, näkyvät skene-näkymässä. Näkymä voi olla joko kolmiulotteinen tai kaksiulotteinen, riippuen siitä, kumpi projektityyppi on kyseessä. Peliobjekteja voi valita ja siirrellä vapaasti näkymän sisällä. (20, s. 19.)

Pelinäkymä

Pelinäkymä simuloi miltä renderöity peli näyttää kohtauksessa esiintyvien kameran tai kameroiden läpi. Työkalurivin play-painiketta painamalla pääsee niin sanottuun ”Play Modeen” eli toistotilaan, jossa voi testata pelin toimivuutta. Tässä tilassa tehdyt muutokset peliin ovat vain väliaikaisia, ja ne nollataan heti, kun poistuu toistotilasta. (22.)

Hierarkiaikkuna

Hierarkianäkymässä luetellaan kaikki aktiivisessa näkymässä, eli skenessä, esiintyvät peliobjektit tekstimuodossa. Jokaisella objektilla on merkintä hierarkiassa, joten skene- ja hierarkiaikkuna ovat linkitettynä toisiinsa. (22.) Hierarkiaikkuna mahdollistaa myös uusien peliobjektien luomisen luo-valikon kautta, ja hakukentän avulla niitä voi etsiä hierarkiasta nimen avulla (20, s. 19).

Unityssä peliobjektit voivat sisältää muita peliobjekteja, joissa pääobjektit ovat niin sanottuja ”vanhempia” ja niiden alle linkitetyt peliobjektit ovat ”lapsia”. Hierarkiaikkuna näyttää nämä suhteet selkeästi sisäistettynä. Kuvassa 4 kaikki peliobjektit, joissa on alasvetovalikko, sisältävät ”lapsi”-peliobjekteja.

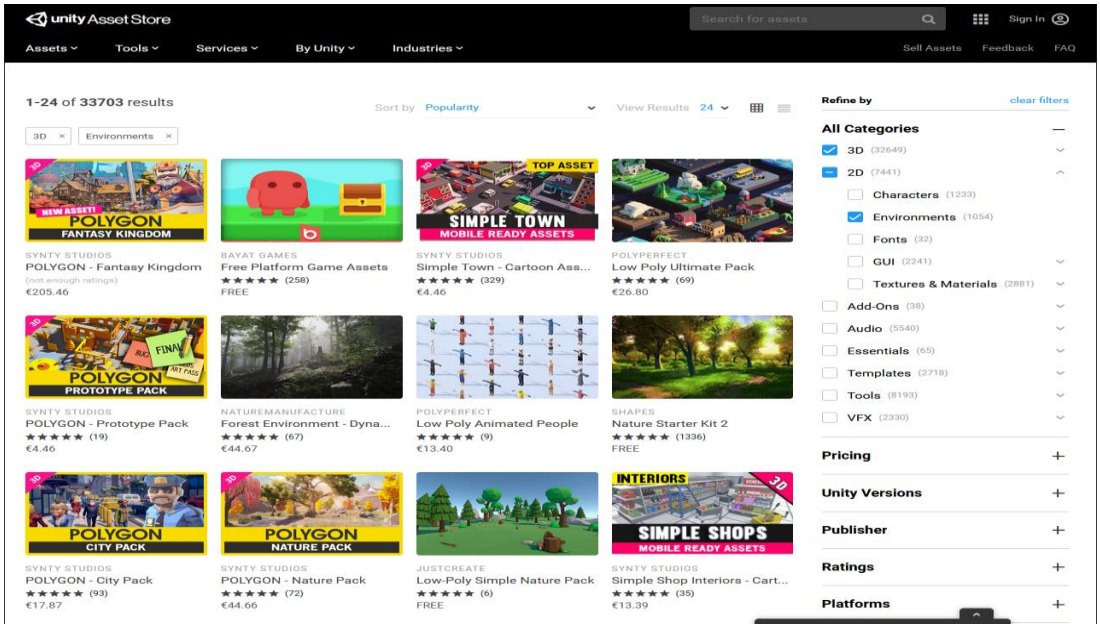
Kuva 4. Hierarkiaikkuna (20, s. 20).

Esimerkki tästä suhteesta voisi olla polkupyörä, joka koostuu runko-, ohjaustanko-, ja rengasobjekteista, missä polkupyörä on ”vanhempi”-peliobjekti ja loput polkupyörän ”lapsi”-objekteja. Se auttaa myös pitämään hierarkian siistinä, sillä monesti toistuvat identtiset peliobjektit voidaan pitää ryhmitettynä. Esimerkiksi kaupunkikohtauksessa käytettävät katuvalo-peliobjektit voidaan hyvin ryhmittää ”katuvalot”-peliobjektiin, joka piilottaa jokaisen katuvalon alasvetovalikon alle. (20, s. 20.)

Unity asset store

Asset Store on verkkokauppa, jossa taiteilijat ja kehittäjät voivat myydä tekemiään tuotteita muiden käytettäväksi. Unity asset store (kuva 5) on samanlainen kuin esimerkiksi Applen ja Androidin sovelluskaupat, mutta sovellusten sijaan sieltä voi ostaa valmiiksi tehtyjä resursseja, jotka voi tuoda suoraan omaan projektiin käytettäväksi (23). Sieltä

löytyy tuhansia ilmaisia ja maksullisia laajennuksia Unity-editoriin, kuten malleja, skriptejä, tekstuureja ja varjostimia. Näiden laajennusten avulla peliyritykset voivat nopeuttaa kehitysaikataulujaan ja parantaa lopullista tuotetta. (20, s. 19.)



Kuva 5. Unity asset store (24).

Tässä projektissa keskityttiin enimmäkseen ohjelmointiin ja pelimaailman luomiseen, joten asset storea käytettiin muihin välttämättömiin asioihin, kuten esimerkiksi tekstuureihin ja 3D-malleihin, joihin tavallisesti tarvitsee 3D-taitelijan avuksi. Resursseja muokattiin tarvittaessa projektiin sopiviksi.

Inspector-näkymä

Kun valitaan näkymästä peliobjekti, inpector-näkymästä voidaan tarkastella ja muokata sen ominaisuuksia. Jokaisella peliobjektilla on uniikit ominaisuudet, joten inspector-näkymä vaihtuu sen mukaan, mikä peliobjekti on valittuna. (22.)

Projektinäkymä

Projektinäkymä näyttää kaikki projektissa olevat resurssit. Tämä paneeli jakautuu kahteen osaan: vasemmalla puolella on hakemistohierarkia, ja kun valitsee yhden kansioista, tiedostoihin sisältyvät resurssit valitusta kansiosta tulevat näkyviin paneelin oikealle puolelle. (21, s. 105.)

Projektinäkymään on hyvä järjestää kaikki projektissa esiintyvät resurssit omiin kansioihinsa niiden tyypin mukaan. Esimerkiksi äänitiedostot, materiaalit, tekstuurit, kohtaukset ja skriptit on hyvä eritellä omiin kansioihinsa. Täten projekti pysyy siistinä ja navigointi on helppoa. (20, s. 20–21.)

5 Peliprojektin suunnittelu

Olen pelannut videopelejä lapsesta lähtien. Muistan kun isäni osti ensimmäisen tietokoneen vuonna 1992 ja vanhat klassikkopelit, kuten Commander Keen ja Bubble Bobble, herättivät 4-vuotiaana kiinnostukseni videopelaamiseen. Siitä asti olen pelannut, enemmän tai vähemmän, nykyhetkeen asti monia erilaisia pelejä. Tämä harrastus vei minut opiskelemaan IT-alaa Metropoliaan ja lopulta erikoistumaan pelisovelluksiin.

5.1 Idean synty

Metropoliassa opiskellessani pelialaa olin ollut tekemässä muutamaa peli prototyyppiä, joten minulla oli jo kokemusta pelien suunnittelusta ja toteutuksesta ennen tätä projektia. En ollut varsinaisesti tehnyt tasohyppelypelejä, ja minua ovat pitkään kiehtoneet indiepeliyrityksien tasohyppelypelit, sillä nykyään mielestäni kaikki pelien innovointi tapahtuu indie-pelien kehityksessä. Suuret pelitalot julkaisevat jo kaupalliseksi todettujen pelien jatko-osia vuodesta toiseen, ja niissä harvoin on mitään uutta ja innovoivaa.

Oman pelin suunnittelu oli erittäin palkitsevaa. Puzzle platformerit ovat aina kiehtoneet, ja otinkin monesta pelistä vaikutteita omaan projektiini. Sellaiset pelit kuin Inside, Little Nightmares ja Limbo ovat tehneet suuren vaikutuksen vuosien varrella, ja päätin jo niitä pelatessa, että haluan tehdä oman tasohyppelyprojektin.

Pelin tyyppi

Ideana oli toteuttaa sivuttaisvierittävä pulmatasohyppelypeli (side-scrolling puzzle platformer) ja samalla ottaa selvää, miten projekti toteutetaan ideasta prototyypiksi. Sivuttaisvierittävä tarkoittaa videopelityyppiä, jossa kamera kuvaa pelaajaa sivusuunnasta ja pelaaja liikkuu näytön vasemmalta puolelta oikealle puolelle kameran seuratessa pelaajan liikkeitä. Hyvä esimerkki ensimmäisistä tämäntyyppisistä peleistä on vuonna 1985 julkaistu Super Mario Bros (kuva 6). Tarkoituksena oli kehittää pelaajalle esteitä, joita hän joutuu selvittämään pelin edetessä, mikä tulee ilmi pelin tyypin sanasta puzzle. Tarkoituksena oli käyttää fysiikkaa pulmien selvittämiseen, niin että pelaaja joutuu fysiikan avulla liikuttelemaan erilaisia esteitä.

Kuva 6. Super Mario Bros -tasohyppelypeli vuodelta 1985 (25)

Tavoite oli kehittää ensimmäinen taso peliin, jossa pelaaja käyttää pelin eri mekaniikkoja pulmien ratkaisemiseen. Siinä olisi vain muutama yksinkertainen puzzle, joka osoittaa pelaajalle, mihin peli kykenee. Kun pelaaja on saanut hyvän käsityksen pelistä, se alkaa vaatia yhä luovempaa ajattelua pelaajalta tavoitteiden saavuttamiseksi.

Pelimoottoriksi valitsin Unityn, koska se on mielestäni yksi markkinoiden parhaista pelimoottoreista ja minulla oli hyvä perusosaaminen siitä. Pelissä pelaajan oli tarkoitus kontrolloida päähahmoa liikkumalla ruudulla joko vasemmalle tai oikealle, kameran seuratessa pelaajaa sulavasti. Halusin implementoida projektiin pelaajan ja ympäristön välille myös interaktiivisuutta, joka käyttää pelin fysiikkaa hyödykseen. Esimerkiksi pelaaja voi olla vuorovaikutuksessa erilaisten vipujen ja nappien kanssa, jotka esimerkiksi avaavat ovia tai erilaisia luukkuja. Projektissa käytettiin Unityn omaa fysiikkamoottoria.

5.2 2.5D-peli

Pelin tuli olla 2.5D-peli. Inspiraationa tähän valintaan toimi Playdead-peliyrityksen kehittämä tasohyppelypeli Inside (kuva 7). Insiden maailmana toimii kolmiulotteinen ympäristö, mutta pelaaja hallitsee hahmoaan kaksiulotteisessa ympäristössä liikkumalla vain eteen- tai taaksepäin. Tästä tulee termi ”2.5D”. Pelaaja ja kamera kuitenkin liikkuvat ennalta määritellyn polun mukaisesti myös z-akselin suuntaisesti, mikä tuo syvyyttä peliin. Tämä mahdollistaa sen, että pelaaja voi nähdä tietyissä kohdissa enemmän ympäristöä kameran zoomatessa poispäin pelaajasta.

Kuva 7. Playdeadin kehittämä puzzle platformer Inside (10).

Vaikka pelaaja voi liikkua vain x- ja y-akselin suuntaisesti, pelin viholliset voivat käyttää z-akselia hyväkseen ja liikkua kolmiulotteisesti hyökätessään pelaajan kimppuun. Tämä antaa 2.5D-peleihin enemmän vaihtoehtoja tasosuunnittelussa verrattuna 2D-tasohyppelyihin.

5.3 Visuaalinen tarinankerronta

Kuva kertoo enemmän kuin tuhat sanaa. Ennen kuin projektin kehitys alkoi, täytyi miettiä tarinaa peliin. Inspiraationa toimi tarinan kehityksessäkin Playdeadin kehittämä Inside-peli, jossa ei ole dialogia eikä varsinaista juonta, vaan pelaaja joutuu itse päättelemään, mitä pelissä oikeasti tapahtuu ja miksi ja mikä on pelin määränpää. Tarkoituksena oli kertoa tarina, jossa pelaajan täytyy itse päätellä pelin juoni ja tehdä oma tulkintansa pelistä sen ympäristön ja tapahtumien kautta. Käytännössä tarinankerronnan tuli olla täysin visuaalinen. Inside-pelissäkin jokainen tekee oman johtopäätöksensä ja tulkintansa tarinasta, eikä mikään johtopäätös ole käytännössä väärä.

Tarina sijoittuu maailmanlopun jälkeiseen maailmaan. Pelin alkaessa pelaaja herää syvältä maanalaisesta ydintuhoa varten rakennetusta holvista. Holvi on tyhjillään, mutta siellä on selvästi ollut muitakin selviytyjiä. Pelaajan tehtävänä on päästä holvista maanpinnalle ja selvittää, mitä on tapahtunut.

5.4 Tunnelma

On vaikeaa määritellä, mikä kuvaa tunnelmaa peleissä. Mielestäni se yhdistää pelin taidetyylin, äänimaailman, tarinankerronnan ja tasosuunnittelun eräänlaiseksi kokonaisuudeksi, joka siten säätelee pelaajan tunnetilaa ja luo immersiota. Halusin luoda projektiin melankolisen ja yksinäisen tunnelman. Ympäristö koostuu isosta tyhjästä maanalaisesta holvista ja hylätyistä rakennelmista. Musiikki on myös iso osa tunnelmaa, joten tarkoituksena oli kehittää musiikkia projektiin.

5.5 Liikkuminen ja interaktiivisuus

Tasohyppelypelien liike sisältää yleensä vähintään juoksu- ja hyppyominaisuuden. Painovoimalla on myös iso merkitys pelattavuuteen (26). Tavoitteena oli implementoida pelihahmon liikkuminen, niin että se reagoi sulavasti pelaajan syötteeseen. Pelissä täytyi pystyä liikkumaan sujuvasti, ja kamera tuli olla implementoitu niin, että se seuraa pelihahmoa sulavasti. Projektiin oli tarkoitus kehittää myös fysiikkaan perustuvia ongelmia.

6 Tasohyppelypelin toteutus

6.1 Projektin aloitus

Insinöörityöprojektissa käytettiin Unity-editorin 2019.1.11-versiota, joka julkaistiin heinäkuussa 2019, ja se oli projektin aloitushetkellä uusin julkaistu versio. Ohjelmointiin käytettiin Microsoft Visual Studio 2019 -ohjelmankehitysympäristöä. Unity-editorin käynnistämisen jälkeen Projects-välilehden New Project -painikkeesta pääsee projektin aloitusnäkymään. Learn-välilehdeltä voi käydä läpi Unityn omia tutoriaaleja ja oppimismateriaaleja, jotka auttavat varsinkin uusia käyttäjiä aloittamaan Unity-editorin käytön. Unity suosittelee näihin tutustumista ennen uuden projektin aloittamista. (22.)

Tämän jälkeen valitaan projektimalli. Projektimallit tarjoavat ennalta valitut asetukset projektien yleisten parhaiden käytäntöjen perusteella. Nämä asetukset on optimoitu 2D- ja 3D-projekteille kaikilla alustoilla, joita Unity tukee. (22.)

Insinöörityöprojektiin valitsin 3D-mallin. Tämän jälkeen projektille täytyy keksiä nimi, minkä jälkeen Unity luo samannimisen kansion, johon tallennetaan kaikki projektiin liittyvät tiedostot. Lopuksi vielä määritellään, mihin tietokoneen kiintolevyllä projekti tallennetaan.

Resurssien tuonti projektiin

Resurssi, eli asset, on mikä tahansa objekti tai kohde, jota voi käyttää projektissa. Resursseja voi tuoda Unityn ulkopuolella luodusta tiedostosta, kuten 3D-mallista, äänitiedostosta, kuvasta tai mistä tahansa Unityn tukemasta tiedostotyypistä. (22.)

Peliprojekteissa on yleensä artisteja, jotka luovat peliin resursseja, mutta tämä insinöörityö keskittyi ohjelmointiin ja kenttäsuunnitteluun, joten resurssit tuotiin muualta. Projektiin löytyi monia hyviä ja ilmaisia resursseja Unityn omasta kauppapaikasta, Unity asset storesta. Ne oli helppo yhdistää projektiin, sillä Unity-editorissa on oma välilehti, josta voi avata kauppapaikan suoraan editorissa. Tuhansien resurssien joukosta löytyy kaikkea 3D-malleista tekstuureihin.

Peliobjektien luominen ja niiden käyttö

Jokainen pelissä esiintyvä esine on GameObject eli peliobjekti. Tämä tarkoittaa, että kaiken, mitä voi ajatella olevan pelissä, on oltava peliobjekti. Se ei kuitenkaan voi tehdä mitään yksin. Sille on annettava ominaisuuksia, ennen kuin siitä voi tulla pelihahmo, ympäristö tai vaikka erikoistehoste. Ne eivät saavuta paljon itsessään, mutta ne toimivat komponenttien säiliönä, jotka toteuttavat todellisen toiminnallisuuden. Esimerkiksi Valo-objekti luodaan liittämällä Valo-komponentti peliobjektiin (22). Tyhjä peliobjekti luodaan nity-editorin GameObject-valikosta valitsemalla create empty. Tyhjällä peliobjektilla ei ole muuta kuin transform-komponentti, josta lisää sivulla 21.

Projektissa käytettiin tyhjiä peliobjekteja lähinnä komponenttien säiliönä. Esimerkiksi maaperä, jossa pelaaja liikkuu, koostuu monesta editorissa luodusta nelikulmiosta, jotka on sijoiteltu peräkkäin, niin että ne muodostavat maanperäkokonaisuuden. Niille luotu tyhjä peliobjekti ”maaperät” on ns. ”vanhempi”, joka pitää sisällään jokaisen ”lapsi”-”maaperäobjektin”. Tämä sallii sen, että tarvittaessa voi valita kaikki nelikulmiot valitsemalla vain tämän peliobjektin.

Prefabit

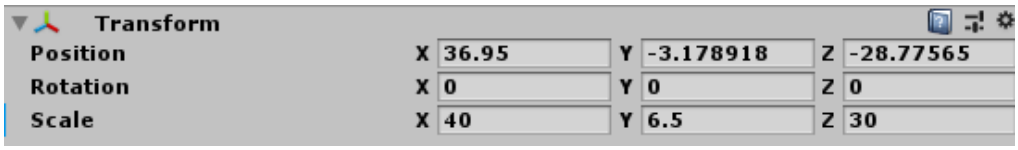
Prefabit ovat erityinen komponenttityyppi, jonka avulla täysin määritetyt peliobjektit voidaan tallentaa projektiin uudelleenkäyttöä varten. Niitä voidaan sitten jakaa kohtausten tai jopa muiden projektien välillä ilman, että niitä on määritettävä uudelleen. Tämä on

varsin hyödyllistä peliobjekteille, joita käytetään monta kertaa, kuten esimerkiksi platformeille. Prefabien suuri etu on, että ne ovat linkitettyjä kopioita projekti-ikkunassa olevista resursseista. Tämä tarkoittaa, että alkupäiseen prefabiin tehdyt muutokset tulevat voimaan myös kaikkiin siitä tehtyihin kopioihin. (22.)

Projektiin tehtiin melkein jokaisesta peliobjektista prefab. Se oli erittäin hyödyllinen esimerkiksi Unityn primitiiveistä tehdystä vivussa, jota käytetään useampaan otteeseen projektissa. Käytännössä prefabista kopioitiin instansseja eri puolille näkymää, missä sitä tarvittiin. Vipu oli aluksi kooltaan turhan iso pelaajaan nähden. Ongelma korjaantui prefabia muokkaamalla, sillä kaikki sen kopiotkin muokkaantuivat oikeankokoisiksi. Jos prefabeja ei käytetä tässä tapauksessa, jokainen vipu olisi täytynyt muokata erikseen, mikä olisi vienyt turhaa aikaa.

Peliobjektin transform

Peliobjektiin on aina liitetty transform-komponentti edustamaan sijaintia ja suuntaa (kuva 8). Sitä ei ole mahdollista poistaa, eikä peliobjektia voi luoda ilman transform-komponenttia. Ne koostuvat position-, rotation- ja scale-valikoista, ja niitä käytetään tallentamaan peliobjektin positio, rotaatio ja skaala.



Kuva 8. Peliobjektin Transform-komponentti.

Position määrittelee peliobjektin sijainnin editorissa. Peliobjektien positiota voi muuttaa Unity-editorissa tai skriptien kautta. Rotationin avulla voidaan kiertää peliobjekteja monin eri tavoin. On mahdollista määrittää kierto maailman tai paikallisakseleilla, eli joko kohtauksen koordinaattien suhteen tai paikallisrotaatio, joka käyttää peliobjektin omaa koordinaattijärjestelmää. Transformia käsitellään 3D-tilassa X-, Y- ja Z-akseleilla tai 2D-tilassa vain X- ja Y-akseleilla. (22.)

6.2 Tasosuunnittelu

Kokeneen tasosuunnittelijan Dave Johnstonin mukaan (27) tasosuunnittelu on aivan liian monipuolinen asia rajoittaa vain yhteen lauseeseen. Yleensä sillä kuitenkin kuvataan peliympäristön luomista, jossa pelaaja on vuorovaikutuksessa pelimaailman kanssa. 1980-luvun alussa tasosuunnittelu olisi ollut prosessi, jossa sijoitetaan esteitä, laattoja ja vihollisia tasohyppelypelin kentälle, kuten esimerkiksi Mariossa tai Sonicissa. 1990-luvulla se eteni 3D-pelaamisen myötä, jolloin tasosuunnitteluun sisältyi lähinnä ”sokkeloiden” luominen pelaajan tutkittavaksi ja vihollisten ja esineiden sijoittaminen oikeisiin paikkoihin. (27.)

Viime aikoina tasosuunnittelusta on kehittynyt paljon monimutkaisempi prosessi, ja siitä on kehittynyt jotain, joka vaatii enemmän kuin koordinaattien ymmärtämisen. Tasosuunnittelu sisältää nyt tyypillisesti valtavan isojen ympäristöjen luomista. Tässä prosessissa kaikki luodaan uudelleen yksityiskohtaisesti, ikkunoista, ovista, rakennuksista ja liikkuvista maisemista. (27.)

Kohtaukset

Kun uusi Unity-projekti luodaan, scene-näkymässä näkyy uusi nimeämätön scene, eli näkymä. Se on tyhjä lukuun ottamatta oletuspeliobjekteja, jotka ovat main camera ja directional light. Main camera, eli pääkamera, näyttää sen, mitä pelaaja näkee ympäristöstä, kun peli on käynnissä. directional light, eli suuntavalo, voidaan ajatella monin tavoin auringoksi, joka valaisee näkymän. (22.)

Kentän rakentaminen näkymissä

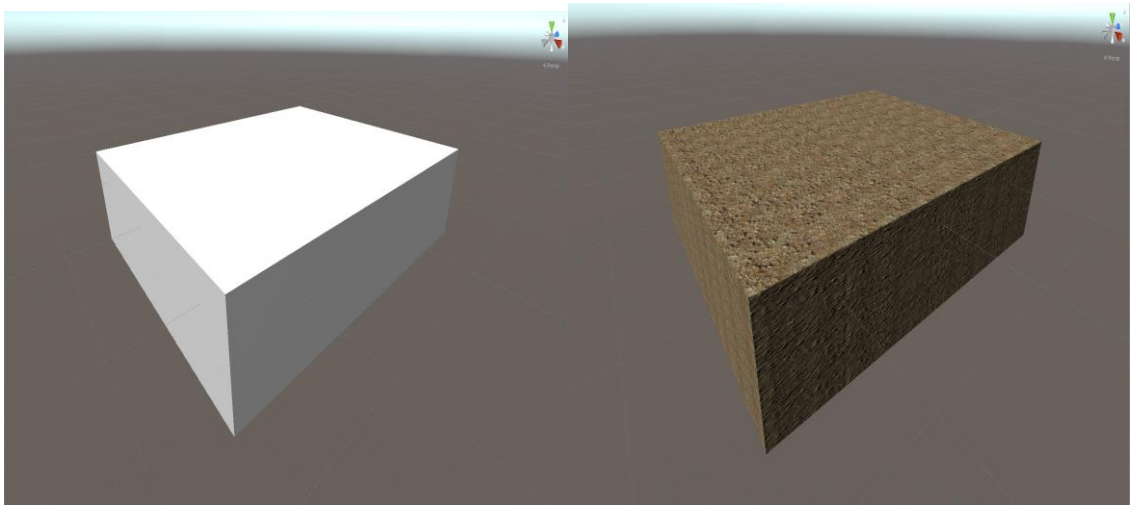
Oli kaksi vaihtoehtoa, miten pelimaailmaa voitiin lähteä rakentamaan peliprojektiin. Ensimmäinen vaihtoehto oli Unityn editorin sisäänrakennettu terrain-työkalu, jonka avulla voi luoda maapohjan peliin ja muokata sitä. Sillä voi suoraan säätää maapohjan korkeutta tai ulkonäköä ja lisätä siihen puita tai ruohoa helposti maalaustyökalun avulla.

Toinen tapa on käyttää Unityn primitiivisiä 3D-objektityyppejä, joita voidaan luoda suoraan Unityssä. Näihin objektityyppeihin kuuluvat kuutio, pallo, kapseli, sylinteri ja taso. (22.)

Projektiin valittiin toinen vaihtoehto. Maaperän ja bunkkerin rakentaminen Unityn omilla kuution muotoisilla peliobjekteilla, joita käyttämällä ja muokkaamalla saa vaivattomasti suunniteltua sisä- ja ulkotiloja. Kuution sivut ovat yhden yksikön pituiset alkuperäisessä muodossaan. Se ei ole alkuperäisessä muodossaan kovin monipuolinen objekti, mutta skaalattuna se on erittäin hyödyllinen seinien, pylväiden ja portaiden ja muiden vastaavien esineiden suunnitteluun. Tämä ratkaisu oli myös paljon parempi suorituskykyä ajatellen. Unity-maastot ovat heightmapeja, kun taas kuutio on Unityn primitiivinen tyyppi, jota voidaan optimoida. heightmap vie enemmän tilaa muistissa, ja sen renderöintiin kuluu enemmän aikaa. Vaikka maapohja olisikin tasainen, Unityn on tarkistettava koko heightmapin korkeus osa kerrallaan, kun taas kuutiosta voi olettaa sen sivujen tasaisuuden (22).

Tekstuurien lisääminen 3D-malleihin

Tekstuurit ovat kuvatiedostoja, jotka asettuvat peliobjektin päälle tai kiertävät sen ympärille antamaan niille visuaalisen efektin. Kuvassa 9 nähdään Unityn kuutio-primitiivityyppi ilman tekstuuria ja tekstuurin kanssa. Tekstuureja asetetaan 3D-malleihin materiaalien avulla. Materiaalit käyttävät erikoistuneita grafiikkaohjelmia, eli varjostimia, joiden avulla tekstuuri renderöidään 3D-mallin pinnalle. (22.)



Kuva 9. Kuutio ilman tekstuuria ja tekstuurin kanssa.

Projektissa maapohjan tuli olla aavikkoa. Unity asset storesta löytyi hyvä tekstuuripaketti, josta löytyi tarvittava aavikkotekstuuri projektia varten. Valmiin materiaalin liittäminen peliobjektiin onnistui vedä ja pudota -periaatteella. Kun materiaali on liitetty, se näkyy peliobjektin inspector-välilehdellä, josta sitä voi muokata. Seuraavaksi sille täytyi määritellä oikeanlainen varjostin. Varjostin on erikoistunut graafinen ohjelma, joka määrittää, kuinka tekstuuri ja valo yhdistetään pikselien luomiseksi renderöityyn peliobjektiin (Unity). Unityssä on nykyään sisäänrakennettuna standard shader, jolla on kattava määrä ominaisuuksia. Sitä voidaan käyttää ”reaalimaailman” esineiden, kuten kiven, puun, lasin, muovin ja metallin renderöintiin. Ennen kuin tämä varjostin julkaistiin, Unityn mukana tuli yli kahdeksankymmentä sisäänrakennettua varjostinta, joista jokaisella oli eri käyttötarkoituksensa. Näitä varjostimia kutsutaan nykyään legacy shadereiksi. (22.)

Ladattujen materiaalien mukana tuli joukko näitä legacy shadereitä, joista aavikossa käytettiin bumbed diffuse -nimistä varjostinta, koska se näytti visuaalisesti hienommalta kuin Unityn Standard-varjostimella saatu tulos. Kuutio-peliobjekteista kehitetyt laatat olivat suuria, ja tämän vuoksi tekstuuri oletusasetuksissaan näytti venyneen objektin päälle. Tähän ratkaisu löytyi materiaalin tiling-valinnasta ja sen X- ja Y-arvoja kasvattamalla. Tämä teki tekstuurista pienemmän, mutta samalla laatoitti sen kuution ympärille niin, että tekstuuri toisti itseään. Tämä sai maaperän näyttämään visuaalisesti paremmalta.

Holvin ulkoasu

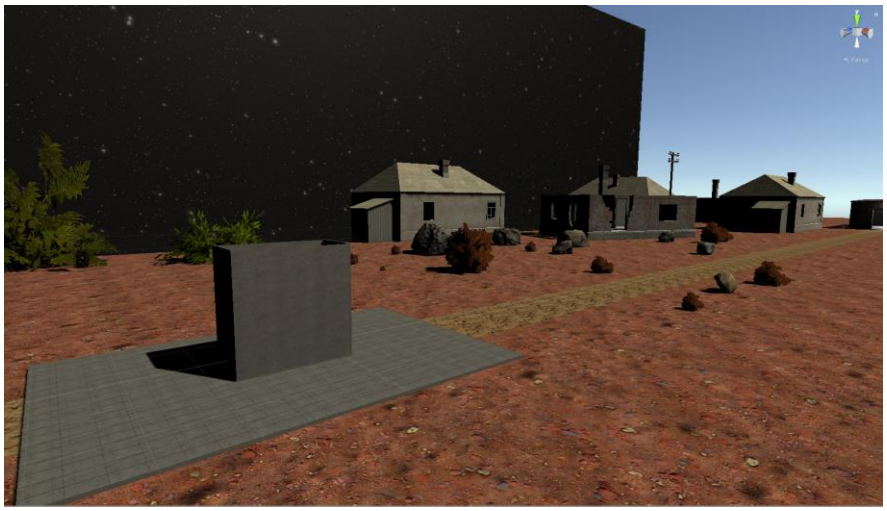
Holvi (kuva 10) suunniteltiin niin, että pelaaja aloittaa pelin sen oikeasta päästä, mistä löytyvät makuu- ja työtilat. Tästä pelaajan olisi tarkoitus saada käsitys, että bunkkerissa on asunut muitakin ihmisiä ja jotain erikummallista on tapahtunut.

Kuva 10. Holvin prototyyppi.

Holvin keskellä on hissi, jonka aktivoimalla pelaaja pääsee ulkomaailmaan. Kun pelaaja liikkuu holvin vasenta puolta kohti, hän tulee ensimmäisenä bunkkerin keskellä sijaitsevan hissin luokse. Hissin vieressä on painike, joka punaisella värillä ja äänimerkillä viestii pelaajalle, että hissi ei ole toiminnassa. Bunkkerin vasemmassa päässä on kasa laatikoita, jotka pelaaja joutuu siirtämään sivuun. Niiden takaa paljastuu vipu, ja sitä painamalla hissin painikkeen valo muuttuu vihreäksi. Tästä pelaaja tajuaa, että hissi on toiminnassa. Bunkkeri rakennettiin Unityn omia primitiivejä käyttäen. Seinät rakennettiin kuutioilla, ja hissin vivussa on käytetty Unityn pallo- ja sylinteriobjekteja.

Ulkotason ulkoasu

Ulkotason layout suunniteltiin niin, että pelaaja etenee kentässä kamerasta katsottuna vasemmalta oikealle. Pelaajan on mahdollista siirtyä kumpaankin suuntaan hissin luota, jolla bunkkerista tullaan maanpinnalle. Vasemmalle mentäessä vastaan tulee iso kallio-lohkare, joka viestii pelaajalle, että sitä ei pysty ylittämään. Pelaajan edetessä oikealle kentässä alkaa tulla vastaan tyhjiä ja hylättyjä taloja. Kuvassa 11 nähdään ulkotason prototyyppi.



Kuva 11. Ulkotason prototyyppi.

6.3 Pelattavuus

Pelattavuus muodostaa suurimman osan pelistä pelaajan näkökulmasta. Esimerkiksi pelimekaniikat, pelisäännöt ja se, kuinka pelaaja hallitsee pelihahmoa ja miten pelaaja voi olla vuorovaikutuksessa pelissä esiintyvien erilaisten esineiden kanssa, ovat kaikki pelattavuuden aspekteja. (28.)

Pelihahmo

Pelihahmo on pelin pelattava päähahmo, jonka toimintaa pelaaja ohjaa tietokoneen näppäimistöä käyttäen. Pelihahmo on aina pelissä prioriteetti, koska sen kautta pelaaja kokee kaiken, mitä pelissä tapahtuu. Pelihahmoksi löytyi hyvä 3D-malli Unity asset storesta. Sen mukana tuli myös animaatioita, joista hyödynnettiin muutamia.

Ohjelmakoodi

Peliobjektien käyttäytymistä ohjataan niihin kiinnitettyjen komponenttien avulla. Unityn avulla voi luoda omia komponentteja ohjelmakoodien eli skriptien avulla. Niiden avulla voi käynnistää pelitapahtumia, muokata komponenttien ominaisuuksia ja vastata käyttä-

jän syötteisiin. Unity tukee C#-ohjelmointikieltä natiivisti, ja sitä käytettiin tässä peliprojektissa. Unityssä voi luoda skriptejä, nimetä niitä ja liittää peliobjekteihin (Unity). Unityssä luodun skriptin mukana ovat vakiona start- ja update-nimiset metodit. Start kutsutaan vain kerran ja heti, kun skripti aktivoidaan. Updatea kutsutaan jatkuvasti, ja se on yleisimmin käytetty funktio pelikäyttäytymisen toteuttamiseksi. (22.)

Liikkuminen pelissä

Liikkuminen on tärkeä osa jokaista peliä. Tasohyppelypelit perustuvat siihen, kuinka pelaaja liikkuu pelimaailmassa. Tämän vuoksi liikkuminen tämäntyyppisissä peleissä on erittäin tärkeää saada reagoimaan sulavasti pelaajan syötteeseen. Projektiin oli tarkoitus saada vasemmalle ja oikealle liikkumisen lisäksi mahdollisuus hyppiä. On tärkeää antaa pelaajan hallita hyppyjen korkeutta tasohyppelypeleissä. Ensinnäkin pelaaja hallitsee hyppyä yhdellä painikkeella. Mitä kauemmin nappia pidetään pohjassa, sitä korkeammalle pelihahmo hyppää. Tämä tuo tavallaan illuusion siitä, että painikkeen painaminen kovemmin saa hahmon hyppäämään korkeammalle.

Jotta pelihahmon sai ylipäätään liikkumaan, siihen täytyi lisätä rigidbody-komponentti. rigidbody-komponentin lisääminen peliobjektiin asettaa sen liikkeen Unityn fysiikkamoottorin hallintaan. Tämän komponentin lisäämisen jälkeen peliobjekti reagoi välittömästi painovoimaan, vaikka siinä ei olisi vielä riviäkään koodia. rigidbody-komponentissa on myös API, jonka avulla voi kohdistaa erilaisia voimia peliobjektiin ja hallita sitä fyysisesti realistisella tavalla. Seuraavaksi pelihahmoon luotiin uusi skripti, johon liikkuvuus implementoitiin koodin avulla. (22.)

Dynaaminen kamera

Insinöörityön pelissä on pääkamera-objekti, jonka kautta pelaaja näkee pelimaailman. Kun luodaan uusi näkymä, Unity lisää siihen pääkameran automaattisesti (22). Projektissa kameraan täytyi luoda skripti ja ohjelmoida siihen toiminnallisuus, jonka avulla se seuraa pelaajaa. Kameran skripti näkyy koodiesimerkissä 1.

Yksinkertaisin tapa olisi kytkeä kamera pelihahmoon, niin että sen sijainti olisi aina keskellä näyttöä. Se saattaa olla hyvä idea testaukseen, mutta se voi todella ärsyttää pelaajaa. Esimerkiksi, jos kamera seuraa jäykästi pelihahmoa jokaisella hypyllä, se voi aiheuttaa helposti pahoinvointia. Se ei ole myöskään kovin sujuvaa, jos kamera seuraa pelihahmoa välittömästi, kun pelaaja alkaa juosta tai pysähtyy äkisti. (29.)

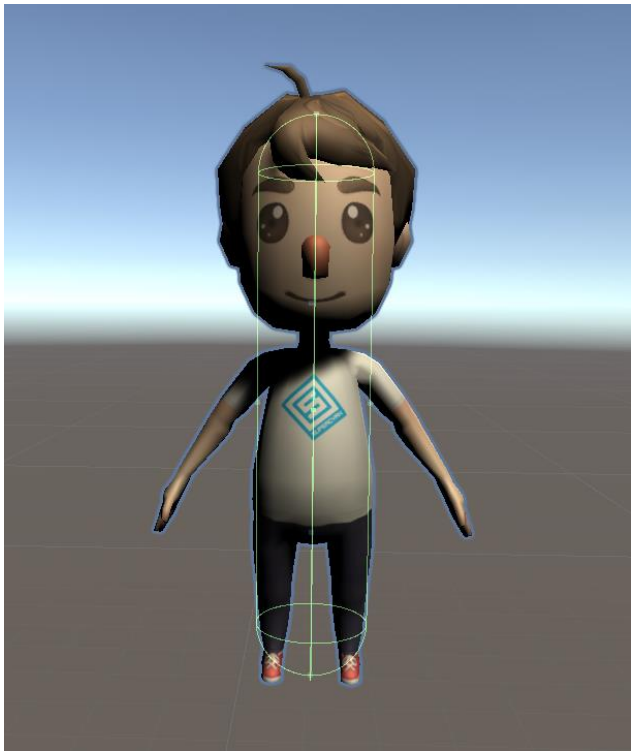
```
    public Transform target;
    public float cameraSmoothing = 5f;

    Vector3 offset;

    void Start()
    {
        offset = transform.position - target.position;
    }

    private void FixedUpdate()
    {
        Vector3 targetCameraPosition = target.position + offset;

        transform.position = Vector3.Lerp(transform.position, targetCameraPosi-
tion, cameraSmoothing * Time.deltaTime);
    }
```

Koodiesimerkki 1. Kameran skripti.

Pelin kameraskriptissä voi valita kohteen, jota kamera seuraa. Tässä tapauksessa se seuraa pelihahmoa. Skriptiin ei ole tallennettu mitään tiettyä välimatkaa objektista, jota halutaan seurata, vaan pelin käynnistyessä start-funktiossa skripti laskee kameran etäisyyden seurattavasta objektista ja tallentaa sen Vector3-muuttujaan. FixedUpdate on funktio, jota käytetään fysiikkaa implementoidessa, ja sitä kutsutaan 50 kertaa sekunnissa. Tämän funktion sisällä määritellään positio, jota kohti kamera liikkuu (targetCameraPosition). Sen arvo tulee pelaajan sijainnista, johon on lisätty alkuperäinen etäisyys. Viimeisessä koodirivissä käytetään Unityn Vector3.Lerp -funktiota, jonka avulla voi interpoloida lineaarisesti kahden pisteen välillä. Se käytännössä mahdollistaa kameran siirtymisen hitaasti sen nykyisestä sijainnista tavoitteeseen, joka muuttuu jatkuvasti kohteen, eli pelihahmon, liikkuessa.

Törmäykset

Seuraavaksi tarvittiin collider-komponentti pelihahmoon. Unityssä collider-komponentit määrittävät peliobjektin muodon fyysisiä törmäyksiä varten. colliderit ovat näkymättömiä muotoja peliobjektien ympärillä, eikä niiden ei tarvitse olla täsmälleen samanmuotoisia kuin peliobjekti itsessään. Yleensä kuitenkin on hyvä käytäntö käyttää collidereita, joiden muoto muistuttaa peliobjektia, johon se liitetään. Kaikkiin peliobjekteihin, joihin halutaan, että pelaaja voi törmätä, on lisättävä collider-komponentti (30, s. 39–40). Kuvassa 12 näkyy projektissa käytetty pelihahmo ja sen capsule-collider-komponentti.



Kuva 12. Pelihahmo ja sen capsule collider.

Yksinkertaisimmat ja vähiten prosessori-intensiiviset colliderit ovat Unityn primitiiviset tyypit, joita löytyy eri muotoisia. Pelihahmoon oli loogisinta liittää kapselin muotoinen collider-komponentti, sillä se sopi hyvin pelihahmon muotoon.

6.4 Animaatiot

Animaatio on prosessi, jossa otetaan staattinen objekti ja idea siitä, kuinka sen tulisi liikkua, ja siihen lisätään tekniikoita, jotta sen saa liikkumaan halutulla tavalla (31). Unityssä voi määrittää animaatioleikkeitä animaatiokomponentille ja hallita toistoa ohjelmakoodista. Animaatioleikkeet järjestetään sitten jäsenneltyyn vuokaavion kaltaiseen järjestelmään, jota kutsutaan animator controlleriksi eli animaattoriohjaimeksi. Se toimii tilakoneena, joka seuraa, mitä animaatioleikettä pitäisi parhaillaan toistaa ja milloin animaatioiden tulisi muuttua tai sulautua yhteen. (22.)

Pelihahmon animaatioiden toteutus

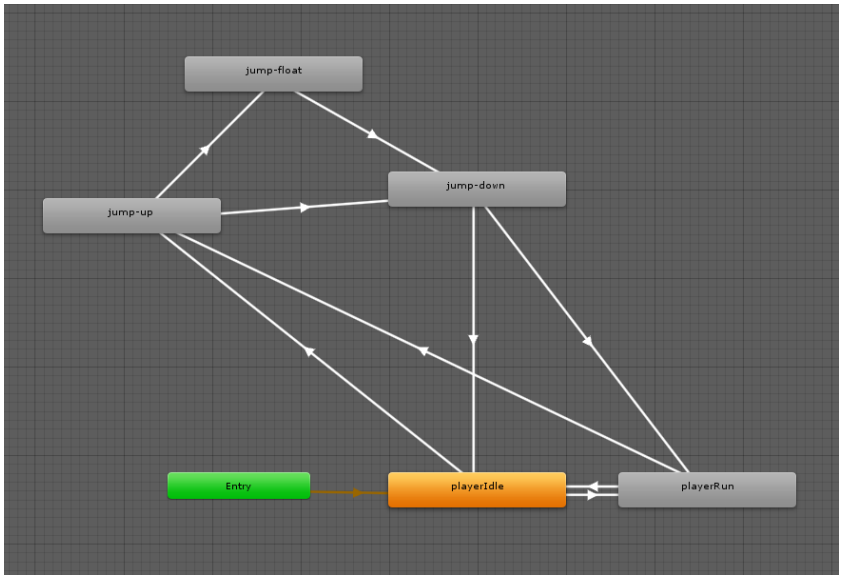
Pelihahmon 3D-mallin mukana tuli animaatioita, joita käytettiin peliprojektissa. Animator controllerin avulla täytyi kehittää animaatioleikkeille oikeat siirtymät ja ehdot. Pelaajan liikkuminen koostui viidestä eri animaatiosta:

- idle-animaatio, joka toistaa pelaajan ollessa paikallaan
- juoksuanimaatio pelaajan liikkuessa vasemmalle tai oikealle
- hyppyanimaatio pelaajan painaessa hyppy-näppäintä
- liitelyanimaatio, joka toistuu kun pelaaja on ilmassa
- laskeutumisanimaatio, joka toistuu, kun pelaaja iskeytyy hypyn jälkeen takaisin maahan.

Animaation siirtymät sallivat animaattoriohjaimen vaihtaa animaatiotilasta toiseen. Siirtymät määrittelevät paitsi sen, kuinka kauan eri tilojen siirtymisen tulisi kestää, myös sen, missä olosuhteissa niiden tulisi aktivoitua. Animaatioiden siirtymisiin voidaan määrittää aktivoitumaan vain, kun siihen määritellyt ehdot täyttyvät. (22.)

Pelihahmon animaattoriohjaimen animaatiotilat näkyvät laatikoina, ja valkoiset nuolet niiden välillä ovat siirtymiä (kuva 13). Nuolen suunta ilmaisee animaatioiden siirtymisen suunnan ja sen, minkä animaatioiden välillä siirtyminen on mahdollista. ”playerIdle” on vakiotila, jossa animaatiokontrolleri on pelin käynnistyessä. Pelihahmon nopeutta kuvaa-

vaa float-muuttujaa käytettiin hyväksi idle-animaation ja juoksuanimaation välillä. Jos nopeusluku on yli tietyn arvon, siirrytään juoksuanimaatioon, ja alle tämän arvon, siirrytään takaisin idle-animaatioon.

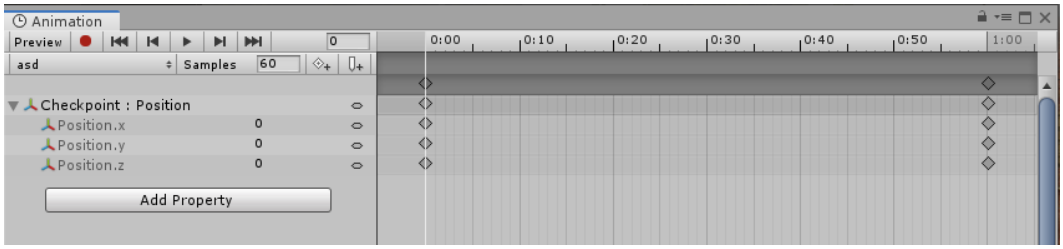


Kuva 13. Pelihahmon animaattoriohjain (animator controller).

Hyppyanimaatiota varten lisättiin jokaiseen objektiin, jonka päälle pelihahmo pääsee, uusi layer nimeltä ”ground”. Layereitä käytetään kaikkialla Unity-editorissa keinona luoda objektiryhmiä, joilla on tiettyjä ominaisuuksia (22). Lisättiin myös Boolen-arvo, joka määrittää, koskettaako pelaaja maata. Tämän jälkeen tehtiin uusi peliobjekti ja sijoitettiin se pelihahmon jalkoihin. Se tarkistaa jatkuvasti, koskeeko pelaaja maata. Jos pelaaja koskettaa maata ja yrittää hypätä, hyppyanimaatio käynnistyy. Hyppyanimaatiosta siirrytään joko liitelyanimaatioon tai laskeutumisanimaatioon, riippuen ilmassa olemisen pituudesta. Kun pelaaja koskettaa maata, Boolen-arvo muuttuu todeksi ja laskeutumisanimaatio suoritetaan. Tästä tilasta siirrytään takaisiin idle- tai juoksuanimaatioon, riippuen siitä liikuttaako pelaaja pelihahmoa tämän laskeutuessaan.

Muiden peliobjektien animointi

Peliprojektin muiden objektien, kuten ovien, vipujen ja hissin liikkuminen, tehtiin suoraan Unity-editorissa. Ensiksi peliobjektille pitää luoda animaatioleike, minkä jälkeen Unity luo sille automaattisesti animaattorikontrollerin. Kuvassa 14 luodaan animaatioleike.



Kuva 14. Animaatioleikkeen luominen.

Useimmissa animaatioissa peliobjektien piti joko liikkua tai pyöriä, eli täten piti käytännössä muuttaa niiden positiota tai rotaatiota ja tallentaa tämä animaatioleikkeeksi. Kaikki peliobjektin tällä hetkellä animoidut ominaisuudet näytetään animaatioikkunan vasemmalla puolella (kuva 14). Ominaisuudeksi valittiin position muuttaminen x-, y- tai z -akselilla. Animaatioikkunan oikealla puolella on aikajana, jonka avulla voi määritellä muun muassa, kuinka kauan animaatio kestää. Tallennus-nappia painamalla pääsee tallennustilaan, jossa voi nauhoittaa animaatioleikkeen.

6.5 Äänijärjestelmä

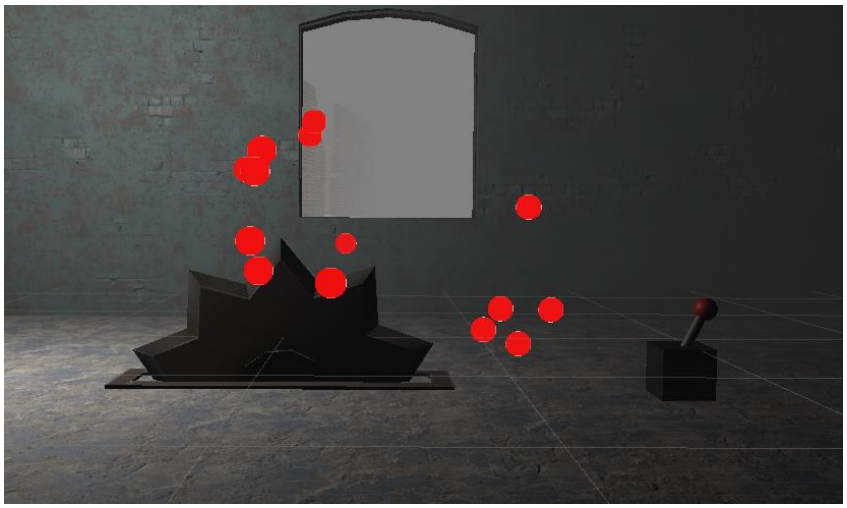
Unityn äänijärjestelmä on joustava ja tehokas. Siihen voi lisätä useimpia äänitiedostomuotoja, ja siinä on hyvät ominaisuudet äänien toistamiseen 3D-tilassa. Unity edellyttää, että äänet ovat peräisin audio source -komponenteista eli äänilähteistä, jotka on liitetty peliobjekteihin. Äänet ottaa vastaan äänikuuntelija, joka oli insinöörityöprojektissa liitetty kameraan. Unity voi sitten simuloida äänilähteen etäisyyden vaikutuksia kuuntelijaobjektista, eli tässä tapauksessa kamerasta, ja toistaa ne käyttäjälle. (22.)

Projektissa käytettiin ilmaisia äänileikkeitä. Internetistä löytyy sivustoja, jotka on tarkoitettu erityisesti ääniefektien jakamiselle. Jotkut sivustoista ovat ilmaisia ja jotkut laskuttavat äänitiedostoista. Youtube-sivusto toimi myös hyvänä äänitiedostojen lähteenä. Projektissa käytettiin hyväksi sivustoja, joiden avuilla voi helposti muuntaa Youtube-videon äänitiedostoksi. Kun video oli muutettu haluttuun äänitiedostomuotoon, se täytyi vielä leikata oikeasta kohdasta, jotta saatiin tarpeettomat äänet ja suhinat poistettua. Tähän tarkoitukseen käytettiin audiotrimmer.com-sivustoa, jonne voi ilmaiseksi ladata omia äänitiedostoja ja muokata niitä. Muokatut tiedostot voi muuntaa haluttuun tiedostomuotoon ja ladata tietokoneelle.

Myös itsetehtyjä äänitiedostoja nauhoitettiin projektia varten. Pelissä esiintyvät juoksu-ja hyppyäänet ovat nauhoitettu Windows 10 ääninauhurilla. Pelihahmon juoksuääni saatiin aikaiseksi tamppaamalla lattiaa. Tätä toistetaan aina, kun pelaaja liikkuu pelissä, ja se päättyy, kun mitään ei paineta. Hyppyäänet nauhoitettiin suulla yhdeksi isoksi tiedostoksi ja siitä leikattiin muutama ääni, joita käytetään pelissä. Nämä tallennettiin ohjelmakoodissa yhteen taulukkoon, ja aina kun pelihahmo hyppää, niistä toistetaan jokin ääni satunnaisesti.

6.6 Partikkeliefektit

Unityssä on hiukkasjärjestelmä, eli Particle System, jolla saa aikaan erilaisia efektejä. Pelaajan kuolemaefekti on kuvattuna kuvassa 15. Se on komponentti, jolla voi simuloida muun muassa nesteitä, pilviä ja liekkejä luomalla ja animoimalla suuren määrän pieniä 2D-kuvia (22). Hiukkanen on mikä tahansa tekstuuri, materiaalinen instanssi tai kokonaisuus, jonka hiukkasjärjestelmä tuottaa. Peliobjekti hallitsee hiukkasjärjestelmää, johon sen komponentti on liitetty. (32.)



Kuva 15. Partikkeliefekti pelaajan kuollessa.

Pelihahmo tuhoutuu terveyspisteiden laskiessa nollaan tai negatiiviseksi luvuksi. Tällöin kutsutaan funktiota, joka deaktivoi pelaajan ja kopioi partikkelijärjestelmän pelaajan sijaintiin saaden aikaan kuolemaefektin. Kuolemaefekti koostuu tehdystä punaisesta materiaalista ja joukosta punaisia pallon muotoisia partikkeleja. Ne lentävät eri suuntiin puolipallon muotoisesti pelihahmon tuhoutuessa.

6.7 Peliobjektien interaktiivisuus törmäyksiä käyttäen

Tässä projektissa oli tarpeen luoda fysikaalista vuorovaikutusta erilaisten peliobjektien kanssa. Tätä varten tarvittiin collider-komponentit jokaiseen peliobjektiin, johon haluttiin pelaajan olevan yhteyksissä.

Laatikot

Ensimmäisessä ongelmassa pelaaja joutuu siirtämään laatikoita hissiä kontrolloivan vivun tieltä. Peliin haluttiin implementoida vuorovaikutus niin, että pelaajan liike hidastuu, jos laatikoita työnnetään tai kannetaan. Unityllä on erilaisia törmäysfunktioita, joista valitsin OnColliderStay-funktion. Sitä kutsutaan jatkuvasti, niin kauan kuin kaksi rigidbodyä tai collideria koskee toisiinsa. Pelaajan liikkuessaan laatikkoa vasten, colliderit koskettaa

toisiaan ja täten pelaajan liike hidastuu, mikä saa aikaan efektin, että laatikko on painava. Laatikoiden kantamiseen tehty skripti on koodiesimerkissä 2.

Projektissa käytettiin Unityn ”tageja” eli tunnisteita hyväksi. Se on viitesana, jonka voi määrittää peliobjekteille. Tunnisteet auttavat tunnistamaan peliobjektit ohjelmointia varten. (22.)

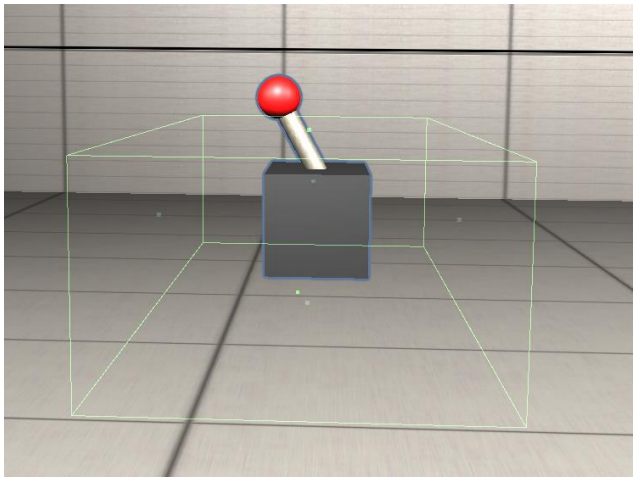
```
private void OnCollisionStay(Collision collision)
{
    if(collision.collider.tag == "Box")
    slowFactor = 2;
    if(Input.GetKeyDown(KeyCode.C) && collision.collider.tag == "Box")
    {
        if (collision.transform.parent != this.transform)
        {
            collision.transform.parent = this.transform;
            canJump = false;
        }
        else
        {
            collision.transform.parent = null;
            canJump = true;
        }
    }

}
```

Koodiesimerkki 2. Laatikoiden kantaminen.

Ohjelmakoodi tarkastaa, onko törmättävän peliobjektin tunniste ”Box”. slowFactor on luku, joka jaetaan pelaajan liikevoimasta. Pelaajan nopeus täten puolittuu työnnettäessä laatikkoa. Jos pelaaja painaa C-näppäintä koskettaessaan laatikkoon, ohjelmakoodi tarkastaa, että pelihahmo ei ole laatikon vanhempi. Jos tämä on totta, laatikosta tulee pelihahmon lapsi. Nyt laatikko on kiinni pelaajassa ja sitä voi kantaa. canJump-Boolen, joka ympäröi pelaajan hyppimisen implementointia, on nyt epätosi, joten pelaaja ei pysty hyppimään laatikon kanssa. Kun pelaaja painaa C-näppäintä uudestaan, if-lause on epätoisi, ja täten laatikko ei ole enää pelihahmon lapsi. canJump taas on tosi, ja slowFactor palautetaan ennalleen. Pelaaja voi siis liikkua ja hyppiä taas vapaasti.

Vipujen toiminnallisuus

Pelissä on kaksi vipua. Toisesta saa holvin hissin toimimaan (kuva 16), ja toisesta aukeaa maan päällä olevan autotallin ovi, samalla aktivoiden saha-ansan liikkuvuuden.



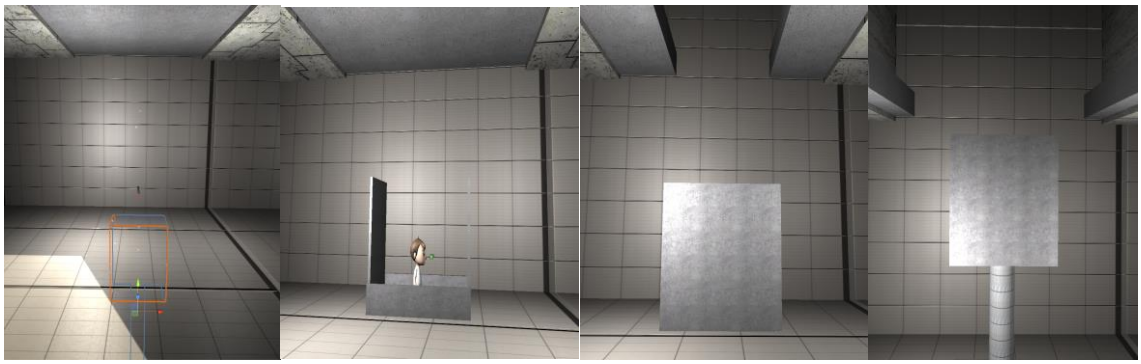
Kuva 16. Holvin vipu ja sen Collider.

Vipu toteutettiin Unityn primitiivimuotoja käyttäen. Se koostuu rungosta ja varresta. Runkona käytettiin Unityn kuutiomuotoa, ja varsi tehtiin sylinteristä ja pallosta. Niissä on iso box-collider-komponentti, jotta pelaaja voi olla vuorovaikutuksessa niiden kanssa vähän etäämmältäkin. Tässä tapauksessa pelaajan ei täydy törmätä collideriin, joten se vaihdettiin triggeriksi. Triggereitä käytetään tapahtumien laukaisemiseen, ja fysiikkamoottori jättää sen huomiotta, joten pelihahmo ei voi törmätä siihen (22). Kummallakin vivulla on oma tunnisteensa, jotta niihin liitetty ohjelmakoodi osaa tehdä oikeat toimenpiteet.

Vivuissa on OnTriggerStay- ja OnTriggerExit- funkiot. Ensimmäinen aktivoituu, kun pelaaja kävelee colliderin alueelle, ja toinen, kun pelaaja poistuu alueelta. Pelaajan ollessa alueella, skripti aktivoi Boolen-arvon todeksi, jolla on mahdollista kääntää vipua. Pelaajan lähtiessä pois alueelta tämä arvo muutetaan epätodeksi, jotta pelaaja ei pysty aktivoimaan vipua liian kaukaa. FixedUpdate-funktio seuraa, painaako pelaaja aktivointinäppäintä Boolen-arvon ollessa tosi. Tämän jälkeen ohjelmakoodi käynnistää vipuun liitetyn animaation ja äänitiedoston ja katsoo tunnisteiden avulla, kumpi vipu on kyseessä. Holvissa oleva vipu kutsuu hissin nappiin liitettyä ohjelmakoodia ja muuttaa siihen liitetyn materiaalin värin punaisesta vihreäksi, mikä viestittää pelaajalle, että hissi on toiminnassa.

Hissin toiminnallisuus

Holvissa sijaitseva hissi koostuu lattiatasosta, neljästä sivuseinästä ja rungosta. Kaikki tehtiin Unityn primitiivimuodoista (kuva 17). Hissiin linkitettiin sarja animaatioita, jotka käynnistyvät pelaajan painaessa hissin nappulaa. Nappulassa on collider ja ohjelmakoodi, joka tarkastaa tunnuksen avulla nappia painettaessa, onko kyseessä pelaaja ja onko napin materiaalin väri vihreä. Jos nämä ehdot toteutuvat, nappulan koodi estää pelaajaa liikkumasta ja aloittaa ”CoRoutinen”. Se on pohjimmiltaan funktio, joka voi keskeyttää sen suorituksen, kunnes annettu ”yield”-toiminto on valmis. Sen avulla voi odottaa määritellyn ajan sekunteina, ennen kuin funktion seuraava rivi suoritetaan.



Kuva 17. Hissin eri vaiheet.

Tässä sekvenssissä on yhteensä viisi animaatiota ja kuusi äänitiedostoa. Hissin noustessa korkeammalle kuilussa kutsutaan funktioita, joka hiljalleen himmentää ruudun mustaksi. Tämä ohjelmakoodi on kiinni kanvaksessa, joka peittää näkymättömänä koko ruudun ja hiljalleen tummentaa sen. Kanvas on alue, jonka sisällä kaikkien käyttöliittymäelementtien tulisi olla (22). Maanpinnalla on kopio hissistä, ja kun ruutu on pimeänä, muutetaan pelaajan ja kameran sijaintia ohjelmakoodissa. Ne siirretään uuden hissin luo, ja tämän jälkeen ruutu muuttuu taas hiljalleen näkyväksi.

Viholliset ja vahinko

Peliin implementoitiin erilaisia ansoja vihollisiksi. Tähän prototyyppiin lisättiin saha- ja piikkiansat, joihin löytyi hyvät 3D-mallit Unity asset storesta. Niihin kehitettiin colliderit,

vahinkoskripti, jolla voi vahingoittaa pelaajaa, ja saha-ansalle toiminnallisuusskripti, johon on implementoitu sen liikkuvuus. Piikkiansa on staattinen objekti maan pinnalla, kun taas saha-ansa liikkuu kahden pisteen välillä. Kuvassa 18 näkyy piikkiansa ja vipu.

Kuva 18. Autotallin prototyyppi.

Pelihahmolla on terveysskripti, jossa määritellään sen maksimiterveys ja jossa seurataan sen terveyspisteiden muutosta sen saadessa vahinkoa. Sen sisältä löytyy funktio, jossa voi parametrillä määrittää tulevan vahingon määrän. Vihollisiin liitetty vanhinkoskripti kutsuu tätä funktiota pelihahmon saadessa vahinkoa. Funktiossa vähennetään pelaajan nykyisestä terveyspisteistä tulevan vahingon määrä. Terveyspisteiden laskiessa nollaan tai negatiiviseksi luvuksi pelihahmo deaktivoidaan ja siirretään viimeisimpään tallennuspisteeseen.

Autotallissa sijaitsevan saha-ansan liikkuvuuden toiminnallisuus määriteltiin siihen liitetyssä skriptissä. Se on aluksi staattinen, ennen kuin pelaaja vääntää vivusta. Vivussa oleva ohjelmakoodi kutsuu sahassa liitettynä sijaitsevaa ja autotallin ovessa olevaa ohjelmakoodia. Saha alkaa liikkua kahden pisteen välillä ja pakottaa pelaajan väistelemään sitä. Samalla autotallin oveen liitetty animaatio aktivoituu ja se aukeaa hiljalleen pakottaen pelaajan väistelemään sahaa.

7 Projektin lopputulos

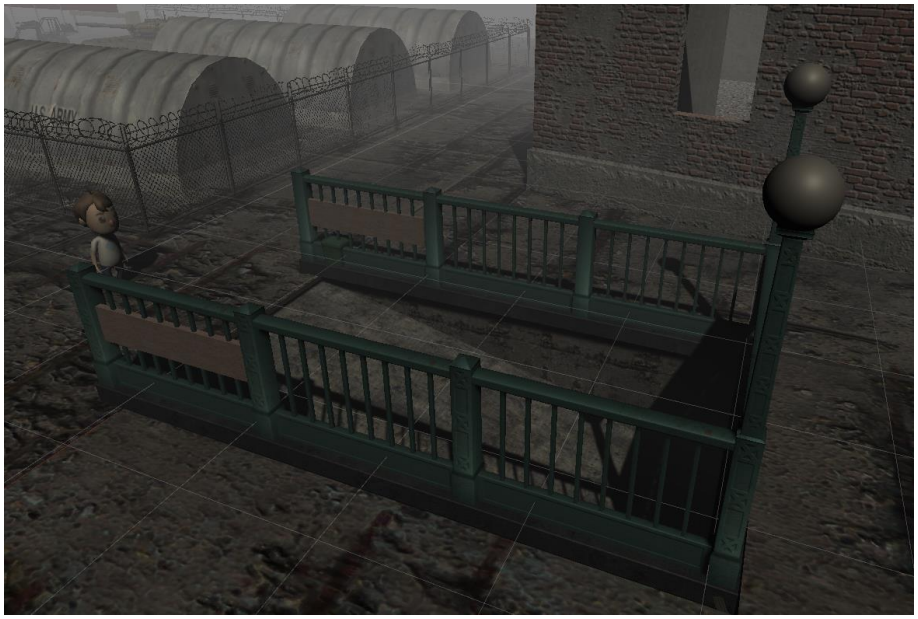
Insinöörityöprojektin loppuvaiheessa peli oli pelattavissa alusta loppuun asti ja kaikki suuret ongelmat oli lähestulkoon korjattu. Kuvassa 19 havainnollistetaan, miltä ulkotason autio kylä näyttää lopullisessa versiossa. Sumu-efekti ja vähäinen valonmäärä tuottavat peliin pahaenteisen tunnelman.

Kuva 19. Pelihahmo hyppäämässä piikkiansan yli.

Pelaajan selviydyttyään aution kylän ansoista, tulee vastaan hylätty armeijatukikohta (kuva 20). Armeijatukikohdan suunnitteluun olisi voinut lisätä enemmän yksityiskohtia, mutta lopputulos oli silti hyvä. Tukikohta koostuu parakeista, ajoneuvoista, piikkilanka-aidasta ja sisäänkäynnistä. Rekvisiittaa löytyi Unity asset storesta ja suunnittelin tukikohdan ulkoasun.

Kuva 20. Hylätty armeijatukikohta.

Peli päättyy, kun pelaaja saavuttaa metrotunnelin sisäänkäynnin (kuva 21). Pelaajan edetessä alas tunnelia pelin ruutu himmenee hiljalleen täysin mustaksi ja peli päättyy. Tarina jatkuu metrotunnelista, jota ei kehitetty tähän prototyyppiin.



Kuva 21. Metrotunnelin sisäänkäynti.

Pelin lopputulokseen voi olla tyytyväinen. Projektiin saatiin toteutettua kaikki suunnitteluvaiheessa keksityt mekaniikat. Prioriteettina oli saada pelihahmon liikkuminen sulavaksi ja responsiiviseksi, mikä olikin yksi projektin vaikeimmista haasteista. Kenttäsuunnittelu oli palkitsevaa, ja peli koostui lopulta kolmesta eri alueesta:

- maanalainen holvi
- autio kylä
- armeijatukikohta ja metrotunnelin sisäänkäynti.

Animaatiot ja pelin äänet saatiin toimimaan synkronoidusti. Pelin äänimaailmaan panostettiin paljon. Tarkoituksena oli tehdä omaa musiikkia peliin, mutta se ei ollut prioriteetti projektissa, joten aikaa ei riittänyt tarpeeksi. Pelissä soiva musiikki on peräisin Carbon Based Lifeforms -nimiseltä artistilta. Sen kappale ”VLA”, joka soi pelin ulkotasossa, sopi hyvin pelin synkkään teemaan. Tämä oli väliaikainen ratkaisu, ja tarkoituksena on tehdä omaa musiikkia peliin tulevaisuudessa.

Fysiikkaan perustuvia pulmia olisi voinut kehittää lisää, mutta niiden suunnittelu oli erittäin aikaa vievää työtä. Pelaajan vuorovaikutus törmäyksien kanssa vaati hienosäätöä, varsinkin laatikoiden kuljettaminen tuotti erilaisia ongelmia, mutta nekin saatiin lopulta toimimaan haluamalla tavalla. Pelaajan vuorovaikutus laatikoiden, vipujen ja vihollisten kanssa toimi kiitettävästi.

Kameraa olisi voinut vielä kehittää, vaikka se saatiinkin sulavasti seuraamaan pelihahmoa. Tarkoituksena kuitenkin oli saada kamera seuraamaan pelihahmoa ennalta määritetyn polun mukaisesti niin, että se liikkuisi myös z-akselin suuntaisesti. Tällä tavoin pelikentästä olisi näkynyt pelaajalle välillä enemmän ympäristöä ja välillä vähemmän, esimerkiksi zoomaamalla ulos pelaajan liikkuessa ulkotason hylätyssä kylässä ja zoomaamalla lähemmäs pelaajan siirtyessä sisätiloihin.

Insinöörityöprojektin pelin kehitys oli mukavan haastavaa työtä. Pelistä tuli melko laadukas, ja tärkeintä on, että sen voi pelata alusta loppuun ilman peliä rikkovia ongelmia. Pelin tekeminen oli niin mukaansatempaavaa, että usein siihen vierähti koko päivä. Peliä tullaan kehittämään tulevaisuudessa ja siitä on tarkoitus kehittää laajempi versio.

8 Yhteenveto

Insinöörityössä pyrittiin tekemään tasohyppelypeli Unity-pelimoottoria käyttäen. Tarkoitus oli tutustua tarkemmin pelinkehitysprosessiin ja samalla ottaa selvää, miten Unityllä voi toteuttaa prototyypin tasohyppelypelistä. Microsoft Visual Studio -työkaluun, jota käytettiin projektin ohjelmakoodin tuottamiseen, oli myös tarkoitus tutustua paremmin projektin edetessä. Ohjelmointi- ja kenttäsuunnittelutaitojen kehittäminen oli myös oleellinen tavoite tässä insinöörityössä. Lopullisena tavoitteena oli saada peli toimimaan niin, että sen voi pelata alusta loppuun asti ilman peliä rikkovia ongelmia.

Projektissa suunniteltiin ja toteutettiin toimiva peli. Siihen tehtiin pelikenttä, joka koostui maanalaisesta holvista ja ulkotasosta. Pelihahmo ohjelmoitiin liikkumaan sulavasti pelikentässä ja projektiin implementoitiin interaktiivisuutta pelihahmon ja erilaisten peliobjektien välille Unityn fysiikkamoottoria ja törmäyksiä hyväksi käyttäen.

Pelien kehittäminen on melko monimutkainen ja pitkä prosessi ilman asianmukaista suunnittelua ja toteutusta. Varsinkin isojen AAA-pelien kehitys vaatii paljon resursseja ja aikaa. Unity on yksi maailman suosituimmista pelimoottoreista pelikehittäjien keskuudessa ja sitä voi suositella sen käyttäjäystävällisen suunnittelun vuoksi. Unity sopiikin tämän vuoksi hyvin tasohyppelypelien kehittämiseen. Sen kattavan käyttöoppaan ansiosta kokemattomatkin kehittäjät saavat kattavaa tietoa Unity-editorista ja sen palveluista.

Insinöörityönä valmistui prototyyppi pelistä, jota ei ole vielä kehitetty loppuun asti. Työtä on tarkoitus hyödyntää pelialalle työllistymisessä. Sen tarkoituksena oli kehittää ammattitaitoa ja antaa siitä näyttöä peliyrityksille työnhakuprosessissa. Tätä projektia kehitetään edelleen, ja tarkoituksena on saada se valmiiksi tulevaisuudessa.

Lähteet

1 Unity Technologies. Verkkoaineisto. <Unity.com>. Luettu 29.4.2020.

2 Peckham, Eric. 2019. How Unity Built the World's Most Popular Game Engine. Verkkoaineisto. <https://techcrunch.com/2019/10/17/how-Unity-built-the-worlds-most-popular-game-engine/>. Luettu 14.10.2020.

3 Grubb, Jeff. 2013. It Takes 600 People for Ubisoft to Make a Triple-A Game on Playstation 4. Verkkoaineisto. <https://venturebeat.com/2013/02/28/it-takes-600-people-for-ubisoft-to-make-a-triple-a-game-on-playstation-4/>. Luettu 26.9.2020.

4 Scully, Ethan. 2020. What is the Game Development Process? Verkkoaineisto <https://careerkarma.com/blog/game-development-process/>. Luettu 10.5.2020.

5 Mitchell, Briar Lee. 2012. Game Design Essentials. John Wiley & Sons.

6 Gregory, Jason. 2009. Game Engine Architecture. Taylor & Francis Group.

7 Kickstarter. Verkkoaineisto. <https://www.kickstarter.com/>. Luettu 2.5.2020.

8 Lowry, Brendan. 2017. The Major Differences Between 'Indie' and 'AAA' Video Games. Verkkoaineisto. <https://www.windowscentral.com/indie-vs-aaa-which-type-game-you>. Luettu 9.5.2020.

9 Caoili, Eric. 2010. IGF Finalist Limbo to Release on XBLA This Summer. Verkkoaineisto. <http://www.gamesetwatch.com/2010/03/igf_finalist_limbo_to_release.php>. Luettu 12.10.2020.

10 Playdead ApS. Verkkoaineisto. <Playdead.com>. Luettu 8.5.2020

11 Ubisoft Entertainment. Verkkoaineisto https://www.ubisoft.com/fi-fi/game/assassins-creed/odyssey. Luettu 22.10.2020

12 Pickell, Devin. 2019. The 7 Stages of Game Development. Verkkoaineisto. <https://learn.g2.com/stages-of-game-development>. Luettu 14.5.2020.

13 Stefyn, Nadia. 2019. How Video Games Are Made: The Game Development Process. Verkkoaineisto. <https://www.cgspectrum.com/blog/game-development-process>. Luettu 20.5.2020.

14 George, Seth. 2019. Phases of the Game Development Process. Verkkoaineisto. <https://gamedevpostmortem.com/phases-of-the-game-development-process/>. Luettu 12.10.2020.

15 Schultz, Charles P; Bryant, Robert & Langdell, Tim. 2005. Game Testing All in One (Game Development Series). Verkkoaineisto. <https://fly-lib.com/books/en/3.36.1.39/1/>. Luettu 9.10.2020.

16 Starloop Studios. 2020. Game Development Stages: How Video Games Are Created? Verkkoaineisto. <https://starloopstudios.com/game-development-stages/>. Luettu 17.10.2020.

17 Perez, Leonard. 2017. Game Production Cycle. Verkkoaineisto. <https://leonardperez.net/game-production-cycle/>. Luettu 15.10.2020.

18 Cypress, Reeves. 2020. How to Playtest Your Game – Game Design Tips. Verkkoaineisto. <https://gamedevacademy.org/game-playtest-tutorial/>. Luettu 21.10.2020.

19 Menard, Michelle & Wagstaff, Bryan. 2014. Game Development with Unity Second Edition. Cengage Learning.

20 Halpern, Jared. 2019 Developing 2D Games with Unity: Independent Game Programming with C#. Apress

21 Eng, Lee Zhi. 2015. Building a Game with Unity and Blender. Packt Publishing.

22 Unity User Manual. Verkkoaineisto. <docs.Unity3d.com>. Luettu 5.5.2020

23 Doran, John P. 2015. Building an FPS Game with Unity. Packt Publishing.

24 Unity Asset Store. Verkkoaineisto. <https://assetstore.Unity.com>. Luettu 3.10.2020.

25 IGN. Verkkoaineisto. <www.ign.com>. Luettu 22.9.2020.

26 Pignole, Yoann. 2014. Platformer Controls: How to Avoid Limpness and Rigidity Feelings. Verkkoaineisto. <https://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_avoid_limpness_and_rigidity_feelings.php>. Luettu 27.9.2020.

27 Johnston, Dave. 2003. What is Level Design? Verkkoaineisto. <https://www.johnsto.co.uk/design/level-design/>. Luettu 9.5.2020.

28 Unreal Engine Documentation. Verkkoaineisto. <https://docs.unrealengine.com/udk/Three/GameplayHome.html>. Luettu 10.10.2020.

29 Heizmann, Jochen. 2017. Camera Logic in a 2D Platformer. Verkkoaineisto. <https://www.gamasutra.com/blogs/JochenHeizmann/20171127/310386/Camera_Logic_in_a_2D_Platformer.php>. Luettu 25.10.2020.

30 Felicia, Patrik. 2003. Getting Started with Unity. Packt Publishing.

31 Animation vs Video Game Design. Verkkoaineisto. <https://www.gamedesigning.org/animation/animation-vs-video-game-design>. Luettu 10.10.2020

32 Unity – The Particle System. Verkkoaineisto. <https://www.tutorialspoint.com/Unity/Unity_the_particle_system.htm.> Luettu 25.10.2020.