Semi Hannuksela

RPG-PELI UNITY3D-PELIMOOTTORILLA

RPG-PELI UNITY3D-PELIMOOTTORILLA

Semi Hannuksela
Opinnäytetyö
Kevät 2018
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä: Semi Hannuksela
Opinnäytetyön nimi: RPG-peli Unity3D-pelimoottorilla
Työn ohjaaja: Veikko Tapaninen
Työn valmistumislukukausi ja -vuosi: Kevät 2018
Sivumäärä: 39

Opinnäytetyön tavoitteena oli tehdä toimiva RPG-pelin prototyyppi. RPG-pelin ideana on saada pelaaja eläytymään pelihahmon taakse edetessään pelimaailmassa. Ensiksi opinnäytetyössä perehdytään RPG-pelien ominaisuuksiin sekä historiaan. Seuraavaksi käydään läpi RPG-pelin toteutus vaihe vaiheelta. Lopuksi on yhteenveto RPG-pelin prototyypin luomisen onnistumisesta.

Projekti toteutettiin Unity3D-pelimoottorilla. Projektin etenemisen aikana hyödynnettiin opetusvideoita, joiden avulla saatiin toteutettua toimivia ominaisuuksia pelille. Myös aikaisempi kokemus Unity3D-pelimoottorin käyttämisestä edesauttoi projektin etenemisessä. Projektin koodikielenä käytettiin C#-kieltä.

Projektin lopussa saatiin tehtyä toimiva RPG-pelin prototyyppi. Projektin aikana saatiin myös hyvä käsitys siitä, miten Unity3D-pelimoottori soveltuu RPG-pelin luomiseen.

Asiasanat: Unity3D, PRG-peli, pelinkehitys

ALKULAUSE

Haluan kiittää ohjaavaa opettajaa Veikko Tapanista opinnäytetyön ohjauksesta sekä Tuula Hopeavuorta kieliasun tarkistamisesta.

13.5.2018

Semi Hannuksela

SISÄLLYS

TERMIT JA LYHENTEET

Animator Controller	Animaattori-ohjaimen avulla voidaan järjestää ja ylläpitää animaatio-leikkeitä. Animaatioiden välille voidaan tehdä siirtymätilat, jotka voidaan tehdä toimiviksi skriptin sisällä.
Asset Store	Verkkokauppa, joka tarjoaa käyttäjien luomia valmismateriaaleja, kuten esimerkiksi tekstuureja, malleja, animaatioita ja valmiita skriptejä tai liitännäisiä.
Collider	Komponentti, joka voidaan lisätä objektille. Sen avulla voidaan luoda törmäystunnisteita muiden objektien kanssa.
Hack and slash	Tarkoitetaan yleensä tietokoneroolipelin pelityyppiä tai pelitapaa. Perustuu lähinnä vihollisten tappamiseen eikä niinkään hahmonkehitykseen taikka juonen käänteisiin.
RPG	Käytetty lyhenne RPG (engl. role-playing game) on videopelien genre, jossa pelaaja ohjaa pelihahmoa edetessään pelimaailmassa.
Scene	Scene eli näkymä sisältää pelin ympäristö-sekä valikkonäkymän
Skriptikieli	Komentosarjakieli eli skriptikieli on kieli, jolla kirjoitetaan komentosarjoja eli skriptejä.
Tile-grafiikka	Tilemapit ovat erittäin suosittu tekniikka 2D-pelin kehityksessä. Grafiikka koostuu pelimaailman tai tasokartan rakentamisesta pineistä, säännöllisesti muotoilluista kuvista, joita kutsutaan laatikoiksi.

1 JOHDANTO

Opinnäytetyö toteutettiin Unity3D-pelimoottorilla. Unity3D on Unity Technologiesin kehittämä monialustainen pelimoottori, jolla voidaan kehittää kaksi- ja kolmeulotteisia selain-, konsoli- ja PC-pelejä. Unityssa pelien ohjelmointiin voidaan käyttää kolmea eri ohjelmointikieltä, JavaScript (UnityScript), Boo sekä C#. Projektin koodikielenä käytettiin C#-ohjelmointikieltä. Koodin kirjoittamiseen käytettiin Microsoft Visual Studiota. (1.)

Opinnäytetyön tavoitteena oli kehittää yksinkertainen ja toimiva RPG-peli käyttäen Unity3D-pelimoottoria. Tavoitteenani oli saada toimivia ominaisuuksia yhteen, kuten esim. animaatiot, collider-ominaisuudet, layout-ominaisuudet sekä vihollisen ja pelaajan liikkuvuus ja toiminnallisuus. Animaatioiden avulla objektien liikuttaminen sekä muut toiminnot saatiin näyttämään realistisimmalta. Layout-ominaisuudella luotiin objekteille elämäpisteet GUI-näkymään. Collider-ominaisuudella lisättiin törmäystunnisteet. Objektien hyökätessä, collider-komponenttien osuessa toisiinsa, vähennettiin elämäpisteitä osujan saaneelta. Tavoitteenani ei ollut saada valmista RPG-peliä, mutta hyvä alku peliprojektille.

Tästä eteenpäin käytän opinnäytetyössäni sanaa ”projekti”, tarkoittaen tällä itse pelin tekemisprojektia opinnäytetyötä varten. Käyttäessäni sanaa opinnäytetyö, viittaan lähinnä kirjalliseen osioon opinnäytetyössä.

2 RPG-PELIT JA HISTORIA

Tietokoneroolipeleistä käytetään usein lyhennettä RPG (eng. role-playing game) tai CRPG (engl. computer role-playing game) (2). Opinnäytetyössäni käytän tietokoneroolipeleistä myös lyhennettä RPG. Tietokoneroolipelit kuuluvat videopelien tyylilajeihin samalla lailla kuin toimintapelit tai strategiapelit. Pelit kuitenkin erottuvat toisistaan niiden sisällön sekä pelimekaniikkojen takia. Nykyajan RPG-peleissä on mukana myös muiden videopelityylilajien pelimekaniikkoja. Tämän takia tietokoneroolipelit yleensä sisältävät paljon toimintaa ja strategiaa, mikä saa pelaajan kiinnostumaan edetessään pelissä. Tietokoneroolipelit eivät kuitenkaan aina ole olleet suosiossa. Tile-pohjaisista roolipeleistä nykyajan tietokoneroolipeleihin suosio on kasvanut kovaa vauhtia.

2.1 Tietokoneroolipelit

Tietokoneroolipelit sisältävät usein monia eri videopelilajityypin pelimekaniikkaa. Näitä yhdistää yleensä se, että niihin on usein liitetty useita eri ominaisuuksia roolipeleistä. Yleensä toiminta-, strategia- tai jopa ammuntapelien mekaanikkoja nähdään RPG-peleissä. Ne tuovat lisää jännitystä peleihin ja sitä kautta pelaajat eläytyvät enemmän oman pelaajahahmonsa taakse. (2.)

RPG-pelit eivät aina ole moninpelejä, vaan pelaaja saa yksin rauhassa edetä omaa matkaansa. Yleensä pelin sisällä on jonkunlainen tarina tai pääjuoni. Monesti tarina tai pääjuoni on pahuuden voittaminen, missä pelaajan tehtävänä on edetä ja estää pahuutta pääsemästä valloilleen. Pelaaja pääsee valitsemaan tälle tielle yleensä oman hahmoluokkansa, joita ovat yleensä esimerkiksi Mage, Paladin, Ranger tai Barbarian. Hahmoluokat ovat keskenään erilaisia. Toiset ovat alussa voimakkaampia fyysisesti, kun taas toiset ovat heikkoja, mutta osaavat tehdä loitsuja. Pelin edetessä hahmoluokat saavat kokemusta tehdyistä tehtävistä ja vihollisten kukistamisesta ja tätä myötä kasvavat voimakkaammiksi ja taitavammiksi. Tämä tuo vaihtelevuutta peliin ja samalla lisää kiinnostusta pelin alusta alkaen. (2.)

Tietokoneroolipeleissä ei kuitenkaan aina pääse eläytymään niin sanotusti rooliinsa. Yksinpelattavissa tietokoneroolipeleissä eläytyminen on melkein mahdotonta, sillä pelaajien välistä interaktiota ei ole. Fallout-peliä voidaan pitää tietokoneroolipelinä, jossa eläytyminen omaan hahmoon vaikuttaa pelin juoneen huomattavasti (2). Myös moninpelattavissa

tietokoneroolipeleissä pelaajien välinen interaktio on tärkeää ja tämän avulla se on lähempänä tavallisia roolipelejä. (2.)

Tietokoneroolipelit ovat kuitenkin yleistyneet viime aikoina. Moninpelien mukana pelaajat pääsevät uppoutumaan hetkeksi omaan tarinanomaiseen maailmaansa ja saavat luoda omat roolinsa.

2.2 Tietokoneroolipelien historia

Roolipelejä on tehty tietokoneille melkein yhtä pitkään kuin tietokoneita on ollut. Tietokoneroolipelien esi-isänä pidetään kuitenkin Ultima I -peliä, ja monille se merkitsee CRPG-genren alkupistettä. Se on Apple II-alustalle tehty peli vuonna 1980, jonka julkaisijana toimi California Pasific Computer Co. (Richard Garriott, alias Lord British). Aikanaan Ultima I -pelin grafiikat olivat ennennäkemättömät, sillä se oli ensimmäinen tile-grafiikkaa käyttävä peli. Seuraavana vuonna 1981 Sir-Tech julkaisi samalle alustalle toimivan Wizardy: Proving Grounds of The Mad Overlord -roolipelin. Ultima-sarjan lisäksi Wizardy-pelit ovat pohjustaneet PC-roolipelien käyttöliittymän. (3.)

Vuosien lisääntyessä pelien sisältöön tuli koko ajan lisää monipuolisuutta sekä niiden toimivuutta ja pelattavuutta parannettiin. The Bird’s Tale sisälsi jo hyvän käyttöliittymän ja värigrafiikat animaatiolla. Liikkuminen niin luolastoissa kuin ulkonakin mahdollistettiin 3D-kuvakulmalla, mitä pidettiin merkittävänä askeleena immersion luomisessa. Immersio viittaa pelaajan kokemukseen syventyä pelaamiseen niin, että hänen keskittymisensä on kokonaan pelissä eikä hän tiedosta pelin ulkopuolista maailmaa (4). Monille tuttu peli Heroes of Might and Magic -pelisarja sai alkuaan Magic-pelisarjan Might And Magic Book I: Secret of The Inner Sanctum 1986 julkaisemasta pelistä. Sen hahmo-ominaisuudet ja mahdollisuus käydä keskusteluja NPC-hahmojen eli pelin sisälle luotujen sivuhenkilöiden kanssa toivat tietokoneroolipeleihin lisää ominaisuuksia ja näin niistä tuli mielenkiintoisempia pelaajille. (3.)

Tietokoneroolipelien kehitys jatkui koko ajan. Grafiikoiden parantuessa voitiin jo pelien sisällä nähdä omien hahmojen taistelut sekä liikkumiset. Seuraava iso askel tapahtui vuonna 1988, kun Wasteland-peli tarjosi avoimen pelimaailman lisäksi tallennusmahdollisuuden. Tämä oli tietenkin merkittävä kehitysaskel tietokoneroolipeleihin, sillä jatkossa pelaaja pystyi jatkamaan peliä siitä, mihin oli jäänyt. (3.)

Aikaisemmissa roolipeleissä oli käytetty aina tile-pohjaista moottoria. Tähän tuli kuitenkin muutos vuonna 1992, kun Origin System kehitti MS-DOS alustalle toimivan Ultima VII -pelin. Tile-pohjaisen moottorin poistuessa, saatiin pelistä näyttävämpi, sillä se mahdollisti entistä paremmat grafiikat (kuva 1). Tuloksena oli ensimmäinen ”maailmasimulaattori”, joka oli siihen mennessä syvin pelikokemus. Pelin pelattavuuteen helpotti myös se, että sitä voitiin pelata hiirellä. Se oli ensimmäisiä tietokoneroolipelejä, jotka oli suunniteltu pelattavaksi hiirellä. (3.)

KUVA 1. Ultima VII mahdollisti paremmat grafiikat, sillä se ei käyttänyt enää vanhaa tile-pohjaista moottoria.

Tästä eteenpäin pelien grafiikat kehittyivät nopeasti ja ensimmäinen peli, jonka pelisarja sai myös minut kiinnostumaan tietokoneroolipeleistä, sai alkunsa 1996 Blizzard Entertainmentin kehittämänä pelinä, Diablo (5). Diablo oli pelityyliltään aikaisempaa yksinkertaisempi ja tärkeimpänä oli vihollisia vastaan taistelu sekä tavaroiden kerääminen. Monien tuntien seikkailujen ja tuhansien vihollisten voittamisen yhteydessä peli toi mukanaan myös toimivan moninpelin. Se antoi mahdollisuuden kavereiden tulla pelaamaan samaan maailmaan ja edetä pelissä yhdessä tuhoten viholliset ja pelastaa pahuuden riivaama maailma. (5.)

2.3 Nykyajan tietokoneroolipelit

Tietokoneroolipelit ovat ajan myötä kehittyneet samanlailla kuin toiminta- tai autopelitkin. Niiden suosio ei kuitenkaan aina ole ollut huipussaan. Monille - niin kuin minullekin - Diablo II -peli oli ensimmäinen mieleen jäävä kokemus hyvästä RPG-pelistä. Se on Bliz-

zard Entertainmentin 29. kesäkuuta 2000 julkaisema Diablo-toimintaroolipelisarjaan kuuluva videopeli. Diablo II -pelin suosio kasvoi niin paljon, että se on yksi kaikkien aikojen myydyin tietokonepeli. Sen tärkein etu oli hyvin tehty Battle.net-verkkopeliyhteisö. (6.)

Diablo II -peliä voidaan verrata nykyajan RPG-peleihin, sillä vielä näinä päivinäkin sadat tuhannet ihmiset pelaavat Diablo II -peliä internetissä. Tietokoneiden komponenttien kehittyminen vaikutti myös pelien grafiikoiden näyttävyyteen. Vuonna 2012 Blizzard Entertainment julkaisi Diablo III -pelin (7). Kahdessatoista vuodessa oli saatu valtavasti kehitettyä peliä niin toiminnallisuuden kuin näyttävyyden kannalta. Peliin oli myös lisätty oma huutokauppa, jossa tavaroita voitiin ostaa, oikealla rahalla taikka pelin sisällä kerättävällä rahalla. Kuitenkin syyskuussa 2013 Blizzard ilmoitti sulkevansa huutokaupan maaliskuussa 2014. Tämä vaikutti niin vahvasti pelaajakuntaan, että sen suosio laski huimasti. Kuitenkin Diablo III -peli on nykyaikanakin hyvä esimerkki hyvästä ja toimivasta RPG-pelistä. Sitä ei kuitenkaan voida täysin luokitella roolipeleihin, jossa pelaaja voisi eläytyä omaan rooliinsa, sillä pelin perusidea on sarjan edellisten pelien tapaan hack and slash -tyylinen. (7.)

Nykyajan tietokoneroolipelejä on kuitenkin useita, eikä Diablo-pelisarja kata kaikkea mitä tietokoneroolipelit yleensä sisältävät. Monin peleissä voidaan kavereiden kanssa viettää yhteistä aikaa, mutta yksinpeleissä pelaaja eläytyy enemmän oman pelihahmonsa taakse. Vuonna 2011 julkaistu The Elder Scrolls V: Skyrim -peli on hyvä esimerkki avoimen pelimaailman pelistä. (8.) Se on lajityypiltään roolipeli ja luokitellaan yksinpeliksi. Siinä pelaaja uppoutuu oman hahmonsa taakse ja etenee eteenpäin tehtävä tehtävältä. Roolipelissä tarinan juoni saa pelaajan uppoutumaan pelin sisälle. Juonen mukana on myös sivutehtäviä, joiden avulla pelaajan hahmo voi esimerkiksi saada lisää kokemusta tai löytää jotain arvokasta. Kun kokemuspisteitä on saavutettu tarpeeksi ja pelihahmo on saavuttanut seuraavan tason, voidaan pelihahmolle lisätä taitopisteitä. Taitopisteitä voivat olla esimerkiksi Agility, Strength, Dexterity ja Stamina. Taitopisteiden avulla pelaajan hahmo kestää enemmän vahinkoa ja pystyy myös tekemään enemmän vahinkoa vihollisille.

Skyrim-peli on tehnyt lisälaajennuksia peliinsä tuomalla erilaisia lisäosia, jotka sisältävät erilaisen tarinan tai jatkoa edelliselle. Lisäosien myötä pelaajamäärä lisääntyy ja näin pelistä saadaan tuottoisampi pelinkehittäjille.

Lisäksi nykyajan tietokoneroolipeleihin kuuluu MMORPG eli massiivinen monen pelaajan verkkoroolipeli. MMORPG-peleissä pelaajalla ei ole yksittäistä päämäärää, vaan tarkoituksena on lähinnä oman hahmon kehittäminen, tehtävien suorittaminen, oman varallisuuden hallinta sekä pelaajien välinen taistelu. Pelit kuten World of Warcraft sekä RuneScape ovat todella suosittuja ja kuuluvat MMORPG-kategoriaan (9).

3 RPG-PELIN TOTEUTUS

3.1 Projektin aloitus

Alussa RPG-pelin suunnitteleminen vei aikaa, sillä pelin sisällön haluttiin rajata heti niin, että siitä ei tulisi liian laaja tai monimutkainen. Erilaisia näkökulmia kokeillessa saatiin hyvä kuva, millainen peli halutaan tehdä. Yleensä RPG-peleihin kuuluu hahmon kehittäminen erilaisilla taitopisteillä pelin edetessä, mutta tässä projektissa ei haluttu keskittyä siihen vaan se lisättäisiin myöhemmin peliin mukaan. Sen sijaan haluttiin panostaa pelihahmon liikkuvuuteen, vihollisen ja pelihahmon väliseen taisteluun, ääniefektien lisäämisen sekä animaatioiden toimimiseen.

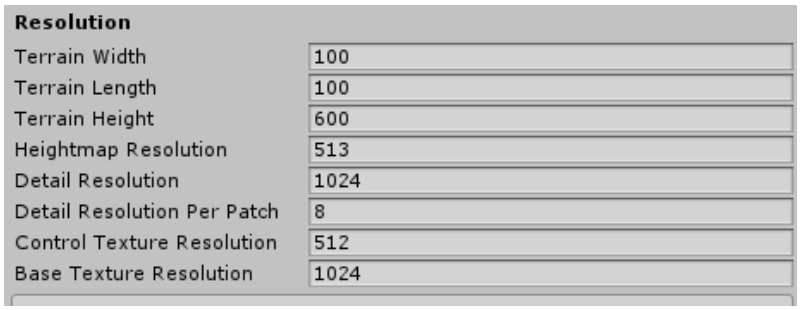
3.2 Pelikentän luonti

Peli aloitettiin tekemällä aluksi pelikenttä Unityllä. Pelikenttä on maastoalue, jonka päällä peliobjekteja liikutetaan. Sen mukana tulleiden asetuksien avulla siitä voidaan tehdä erittäin näyttävä. Sen lisääminen tapahtuu valitsemalla yläikkunasta **GameObject -> 3D Object -> Terrain**. Kun Terrain-objekti on lisätty, voidaan sen muokkaaminen aloittaa.

Ensimmäiseksi muutetaan Terrain-objektin kokoa. Sen muokkaaminen tapahtuu Terrain-objektin lisäämisen yhteydessä tulleen Terrain-työkalun avulla. Tämän kautta päästään muokkaamaan Terrain-objektin asetuksia (kuva 2).

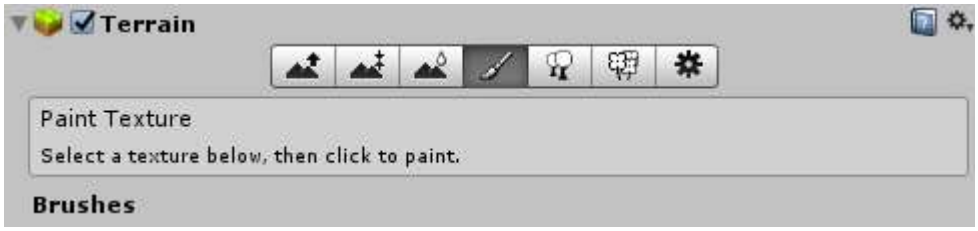
KUVA 2. Terrain-työkalun Terrain asetukset

Sieltä voidaan muokata resoluution ominaisuuksia. Sen avulla voidaan muokata Terrain-objektin leveyttä, pituutta sekä korkeutta. Se mahdollistaa myös yksityiskohtien tarkkuuden määrittelemisen (kuva 3). Tämä on hyvä tehdä alussa, jos pelikentästä haluaa pienemmän, sillä sen muuttaminen myöhemmin voi vaikuttaa pelikenttään tehtyihin materiaaleihin.



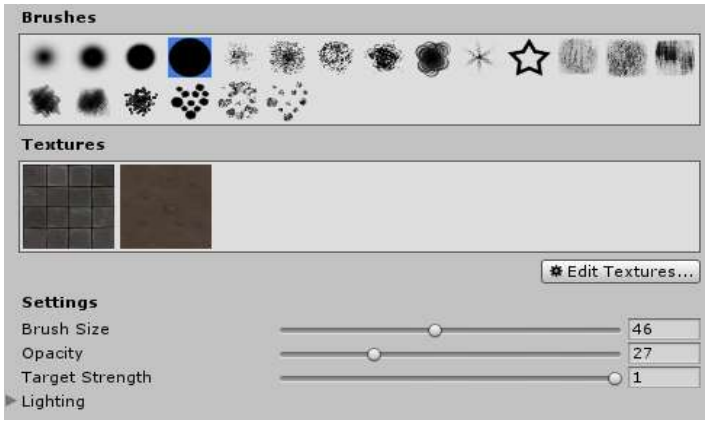
KUVA 3. Terrain-objektin tarkkuuden määritteleminen

Kun Terrain-objektin koko on saatu muokattua, lisätään maastolle tekstuuri. Tätä varten ladattiin Asset Storesta sopivat tekstuurit. Sitten siirrytään Terrain-työkalun tekstuurin maalausasetuksiin (kuva 4).



KUVA 4. Terrain-työkalun välilehti tekstuurin lisäämiselle

Täältä pelikenttään lisätään kaksi uutta tekstuuria. Sen lisääminen onnistui avaamalla **Edit Textures** -valikko. Sen jälkeen valitaan tekstuuri, jota halutaan käyttää maaston maalaamiseksi. Maalaamiseen voidaan käyttää erilaisia siveltimiä sekä siveltimen muotoa ja kokoa voidaan muuttaa (kuva 5).

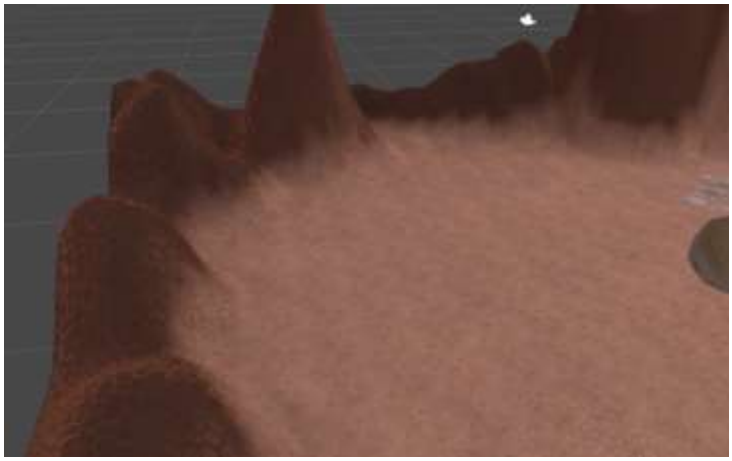


KUVA 5. Terrain-työkalun tekstuurin lisäämisasetukset

Kun Terrain-objektin tekstuurit on muokattu, voidaan aloittaa maaston korkeuden muokkaaminen. Maaston korkeutta muokataan Terrain-työkalun ensimmäisestä välilehden vaihtoehdosta (kuva 6).

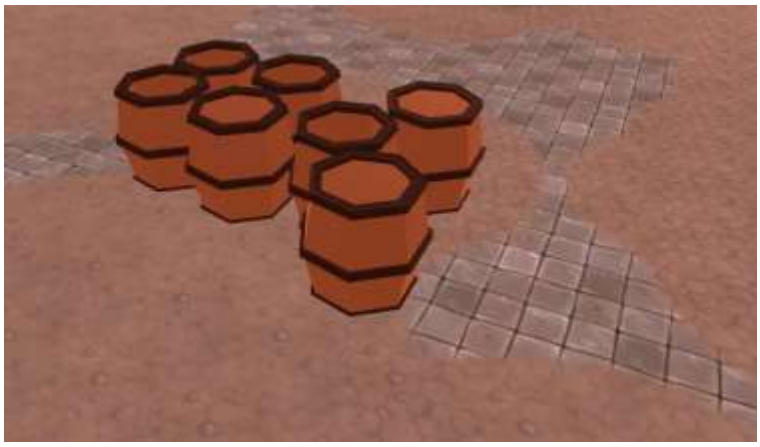
KUVA 6. Terrain-työkalun maaston korkeuden muokkaamisen välilehti

Klikkaamalla pelikentän maasto-aluetta voidaan nostaa maaston korkeutta. Kun maasto halutaan saada tasaisemmaksi, saadaan se tehtyä pitämällä näppäimistöstä Shift-näppäintä pohjassa yhtäaikaisesti, kun klikataan hiirellä aluetta, jota haluttaan tasata. Tämän avulla muokattiin pelikentän reunat niin, että sitä ympäröi korkeat vuoret (kuva 7).



KUVA 7. Pelikentän reunoille tehdyt vuoret

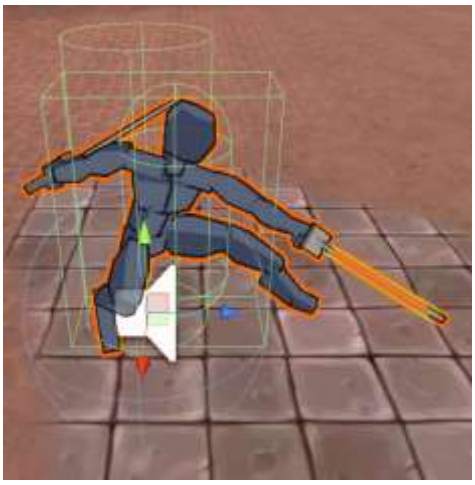
Lisäämällä lopuksi pelikenttään muutama objekti saadaan pelikentästä näyttävämpi. Materiaalien lataamisen jälkeen voidaan objektit siirtää pelikenttään (kuva 8).



KUVA 8. Esimerkki pelikenttään lisätystä tynnyri-objektista

3.3 Pelihahmon lisääminen

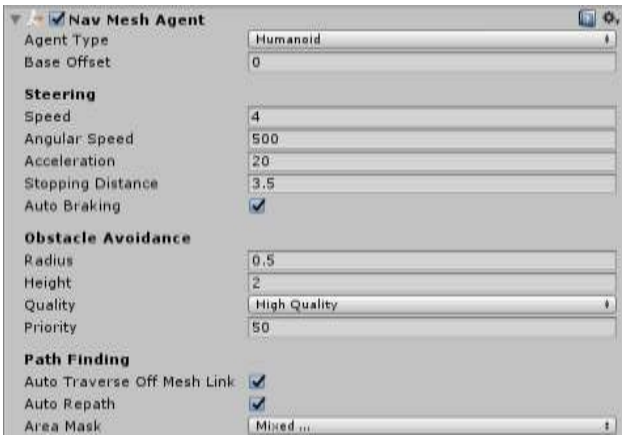
Projektissa seuraavaksi aloitettiin pelihahmon tekeminen. Pelihahmon tekemiseen voidaan ladata valmis paketti Asset Storesta. Siten saadaan valmis pelihahmo, joka sisältää valmiiksi kolme animaatiota. Animaatiot sisältävät Idle-, Attack sekä Run-animaatiot. Lataamisen jälkeen pelihahmon elementti siirretään pelikenttään (kuva 9).



KUVA 9. Asset Storesta ladattu Ninja-pelihahmo

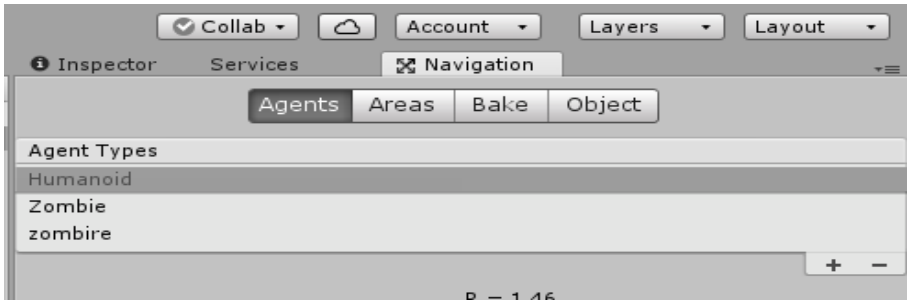
3.3.1 NavMesh Agent

Pelihahmon lisäämisen jälkeen pitää pelihahmo saada liikkumaan pelikentällä. Tämä aloitetaan lisäämällä pelihahmolle NavMesh Agent -komponentti. Sen avulla pelihahmolle voidaan asettaa nopeus, kiihtyvyys, kulmanopeus sekä pysähtymisetäisyys (kuva 10).



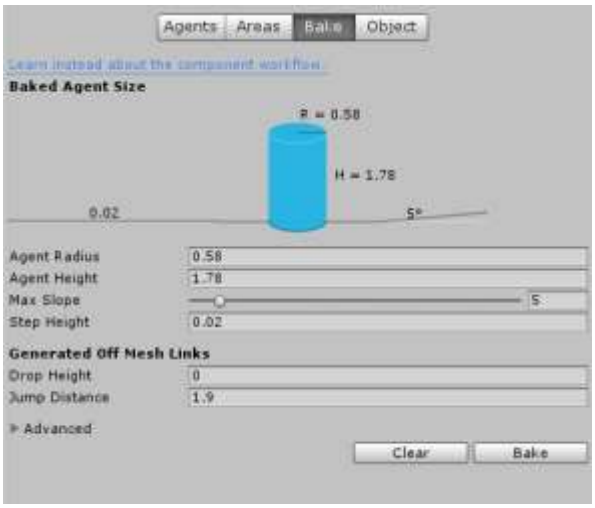
KUVA 10. NavMesh Agentin lisääminen pelihahmoon

NavMesh Agent -komponentti tuo mukanaan myös reitin etsimisominaisuuden eli navigoimisen. Tämän voi tehdä valitsemalla yläikkunasta **Window -> Navigation** -vaihtoehto. Tämän jälkeen navigointi asetukset aukeavat omaan välilehteensä (kuva 11).



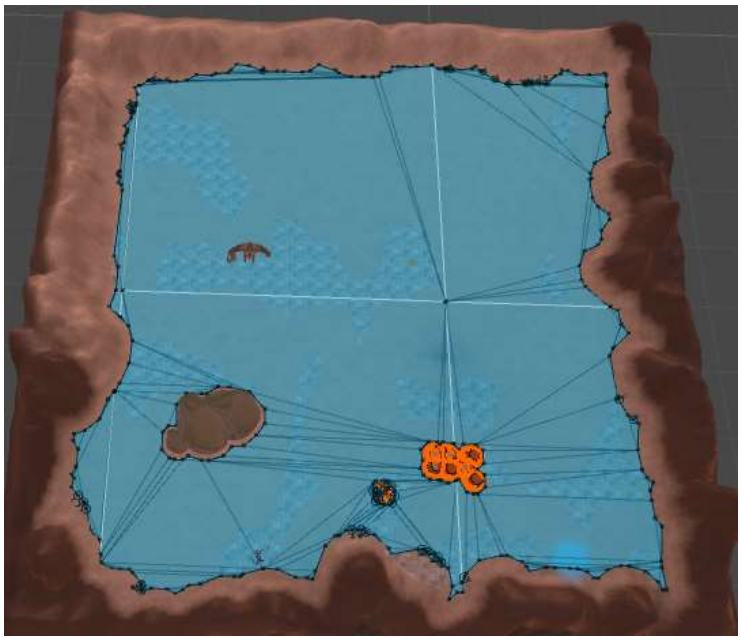
KUVA 11. Navigointi-välilehti

Tämän sisällä voidaan määritellä NavMesh Agent -asetukset. Agentin tyyppejä voidaan luoda useampia. NavMesh Agent -komponentin lisättyä agentin oletustyyppi on nimeltään Humanoid. Tämän jälkeen määritellään korkeus, säde, askelkorkeus sekä maksimi pinnankorkeus. Askelkorkeudella sekä pinnankorkeudella määritellään, kuinka korkeiden esteiden päälle pelihahmolla voidaan mennä tai kuinka korkeiden esteiden yli voidaan kävellä. Tämän jälkeen määritetään Bake-asetukset. Ensimmäiseksi määritellään säde ja korkeus. Projektissa tämän määrittelemiseen käytetään apuna pelihahmolle lisätyn NavMesh Agent -komponenttiin asettamia tietoja. Bake-ominaisuuden säde ja korkeus asetetaan hiukan isommiksi. Tämän jälkeen asetetaan askelkorkeuden sekä pinnanmaksimin korkeus. Nämä asetetaan myös todella vähäiselle, sillä pelikenttää ympäröivien rinteiden ylipääsemistä ei haluta mahdollistaa. Tämän jälkeen, kun kaikki ominaisuudet saadaan asetettua, voidaan Bake-asetuksien sisältä ajaa Bake-toiminto (kuva 12).



KUVA 12. Bake-asetukset ja Bake-näppäin

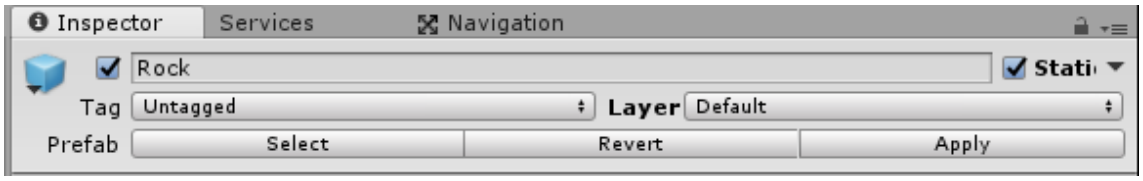
Bake-toiminnon suoriuduttua voidaan katsoa, mitkä alueet pelikentässä mahdollistavat pelihahmon liikkumisen. Rajatulla vaaleansinisellä alueella pelihahmon on mahdollista liikkua (kuva 13).



KUVA 13. Pelikenttä ja alue, jossa pelihahmo voi liikkua

Tämän yhteydessä tarkastetaan kaikki objektit, jotka on lisätty pelikenttään. Objektien lisäämisen yhteydessä ne asetetaan static-tilaan eli staattiseen tilaan (kuva 14). Kun valitut objektit on asetettu staattiseen tilaan, täytyy Bake-toiminto suorittaa uudestaan.

Tämän jälkeen huomataan, että objekteja ympyröi rajaviivat, jotka estävät niiden läpikävelemisen. Jos haluttujen objektien läpikäveleminen halutaan kuitenkin mahdollistaa, pitää objekti poistaa static-tilasta ja Bake-toiminto suorittaa uudestaan.



KUVA 14. Rock-objekti asetettu static-tilaan

3.3.2 Elämä- ja manapisteet

Yleensä RPG-pelissä pelihahmolla on alussa tietty määrä elämäpisteitä. Elämänpisteitä menetetään matkan varrella taistellessa vihollisia vastaan. Kun elämäpisteet loppuvat, pelihahmo kuolee ja peli joudutaan aloittamaan alusta. RPG-pelihahmolla voi olla myös manapisteitä. Näitä käyttäen pelihahmolla voidaan tehdä jokin erikoishyökkäys, joka kuluttaa tietyn määrän manaa. Projektissa manapisteitä menettää joka kerta, kun pelihahmolla tehdään attack-animaatio. Ajan kuluessa manapisteitä kuitenkin kertyy takaisin. Projektissa elämä- sekä manapisteet toteutetaan käyttäen Unityn GUILayout-luokkaa.

Tämä aloitetaan lisäämällä pelihahmolle uusi C#-skripti. Elämäpisteille sekä manalle annetaan maksimiarvo sekä tämän hetkinen arvo. Molempien arvoiksi asetetaan sata. Tämän jälkeen OnGUI-metodin sisällä lisätään neljä uutta laatikkoa GUI-näkymään (kuva 15).

```
private void OnGUI()
{
    GUI.Box(new Rect(5, 30, 80, 20), "HP");
    GUI.Box(new Rect(5, 50, 80, 20), "Mana");

    GUI.Box(new Rect(85, 30, BarLength, 20), CurrentHealth.ToString("0") + "/" + MaxHealth);
    GUI.Box(new Rect(85, 50, BarLength, 20), CurrentMana.ToString("0") + "/" + MaxMana);

}
```

KUVA 15. Laatikoiden lisääminen GUI-näkymään

Seuraavaksi lisätään AdjustCurrentHealth-metodi missä asetetaan ehto, että elämäpisteet eivät voi mennä yli annetun maksimiarvon. Tämän lisäksi elämäpisteiden arvo ei voi mennä alle nollan (kuva 16).

```
public void AdjustCurrentHealth(int adj)
{
    CurrentHealth += adj;

    if (CurrentHealth >= MaxHealth)
    {
        CurrentHealth = MaxHealth;
    }

    if(CurrentHealth <= 0)
    {
        CurrentHealth = 0;
    }

}
```

KUVA 16. AdjustCurrentHealth-funktio

Sama tarkistus tehdään myös mananarvon kohdalla, mutta sen arvoa kasvatetaan koko ajan (kuva 17).

```
if(CurrentMana >= 0)
{
    CurrentMana += Time.deltaTime;
}
```

KUVA 17. Mananarvon lisääminen

Tämän jälkeen elämä- sekä manapisteet näkyvät pelin vasemmassa ylänurkassa (kuva 18).

KUVA 18. GUI-näkymä elämäpisteistä sekä manasta

3.3.3 Pelihahmolla liikkuminen

Pelin sisällä pelihahmoa voidaan liikuttaa hiiren avulla. Klikatessa hiiren vasenta näppäintä pelikentän sisällä saadaan pelihahmoa liikutettua kyseiseen kohtaan. Tämä toiminnallisuus aloitetaan lisäämällä pelihahmolle uusi C#-skripti.

Kun pelihahmoa halutaan saada liikutettua hiiren avulla, täytyy käydä läpi Physics.Raycast-luokkaa. Luokan avulla luodaan säde kameran sekä maaston välille. Tähän löytyy paljon opetusmateriaalia sekä opetusvideoita, jotka helpottavat sen ymmärtämistä.

Physics.Raycastin käyttämisen yhteydessä sille täytyy määritellä tarvittavat parametrit. Parametrien määrittelemisen yhteydessä asetetaan origin eli aloituspaikka sekä direction eli suunta. Aloituspaikka saadaan selville käyttämällä apuna pelin sisällä toimivaa kameraa. Kameran läpi kulkee kuvitteellinen Raycast eli ”lasersäde”, kunnes se törmää collider-komponentin kanssa. Kameran läpi menevä säde ottaa vastaan hiiren olinpaikan. Tämän jälkeen säteen tiedot palautetaan sen osumakohdasta. Säteen törmäyspiste saadaan selville käyttämällä RayCastHit-rakenteen Point-ominaisuutta. Tähän kirjoitetaan uusi funktio nimeltä MoveToPoint, joka ottaa vastaan Vector3 point -parametrin. Point palauttaa törmäyspisteen käyttämällä x, y, z-koordinaatistoa. Tämän jälkeen pelihahmon määränpääksi asetetaan törmäyspisteen koordinaatit (kuva 19).

```
public void MoveToPoint(Vector3 point)
{
    agent.SetDestination(point);
}
```

KUVA 19. MoveToPoint funktio

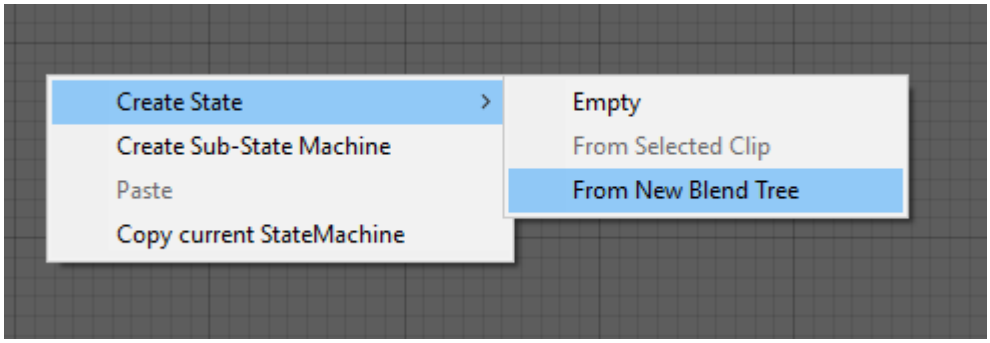
Tämän jälkeen parametriksi asetetaan maksimietäisyys. Tällä tarkoitetaan etäisyyttä, jossa säde tarkistaa törmäykset. Tähän asetetaan 100, jolloin rajataan mahdollisuus liikkua hiirellä koko pelikentän alueella. Viimeiseksi parametriksi asetetaan LayerMask. LayerMaskin avulla voidaan valikoivasti sivuuttaa törmäyskohteet, johon lasersäde osuu. LayerMask asetetaan skriptin alussa julkiseksi, jonka jälkeen pelikentän tasoja voidaan lisätä mahdollisiksi törmäyskohteiksi. Tässä säteen törmäyskohdaksi halutaan asettaa pelikentän maastoalue. Maastoalueelle lisätään uusi Layer eli taso ja sille annetaan nimeksi Ground. Tämä lisätään ainoaksi mahdolliseksi säteen törmäyskohteeksi. Kun kaikki tarvittavat parametrit on määritelty, saadaan pelihahmoa liikutettua (kuva 20).

```
public LayerMask movementMask;
public Interactable focus;

PlayerGUI playergui;
PlayerMotor motor;
Camera cam;

private void Start()
{
    playergui = GetComponent<PlayerGUI>();
    cam = Camera.main;
    motor = GetComponent<PlayerMotor>();
}

private void Update()
{

    if(Input.GetMouseButtonDown(0))
    {

        Ray ray = cam.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if(Physics.Raycast(ray, out hit, 100, movementMask))
        {
            motor.MoveToPoint(hit.point);

            RemoveFocus();
        }


    }
```

KUVA 20. PlayerMovement skripti

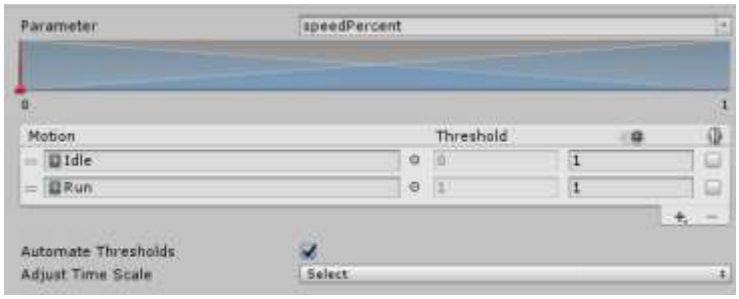
3.3.4 Animaation lisääminen pelihahmon liikkuessa

Animaation lisääminen yhtäaikaisesti, kun pelihahmoa liikutetaan, saadaan toimimaan muokkaamalla Animator-komponentin ikkunatilaa. Tämä aloitetaan lisäämällä uusi Blend Tree, joka saadaan painamalla Animator-ikkunan sisältä hiiren oikeanpuoleista näppäintä ja valitsemalla **Create State -> From new Blend Tree** (kuva 21).



KUVA 21. Blend Treen lisääminen

Blend Treen avulla on helppo sekoittaa kaksi tai useampi animaatio, jotka ovat lähellä toisiaan. Hyvänä esimerkkinä voidaan pitää kävelyanimaation sekä juoksuanimaation sekoittamista hahmon liikkumisnopeuden mukaan. Blend Treen lisäämisen jälkeen kaksoisklikkaamalla saadaan avattua Blend Treen kaaviokuva näkymä. Tämän jälkeen klikkaamalla kaaviokuvan Blend Treetä voidaan se nimetä. Nimeämisen jälkeen sille lisätään yksi float-parametri, speedPercent. Sen jälkeen lisätään kaksi uutta animaatiota, idle-animaatio sekä juoksuanimaatio. Animaatioille voidaan määrätä myös vaihtumisnopeudet, jotka asetetaan molemmat arvoon 1. Animaatioiden raja-arvot asetetaan automaattisiksi (kuva 22). Raja-arvoja vaihtumistilat määritellään skriptin sisällä.



KUVA 22. Blend Treen animaatioiden lisääminen sekä niiden asetukset

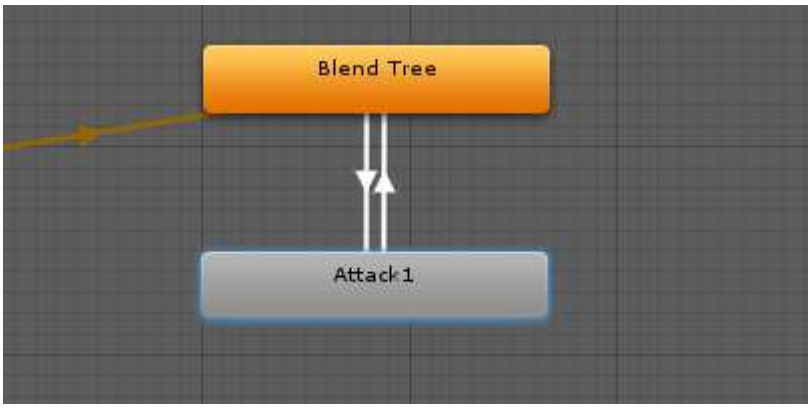
Skripti aloitetaan viittaamalla aluksi NavMesh Agent -komponenttiin sekä Animator-komponenttiin. NavMesh Agentin avulla on aikaisemmin määritelty pelihahmolle nopeudet. Näitä hyödyntäen luodaan Update-metodin sisälle uusi float-muuttuja. Muuttujan arvoksi asetetaan pelihahmon tämän hetkinen nopeus, joka jaetaan pelihahmon maksiminopeudella. Tällä saadaan arvo väliltä 0 ja 1. Animator-komponentin viittaaminen alussa mahdollistaa sen sisältämän speedPercent-parametrin asettamisen. Koska parametrin arvoksi on asetettu float-tyyppi, täytyy asettaa float-arvot. Ensimmäiseksi float-arvon parametriksi annetaan Animator-ikkunan sisältämä parametri. Toiseksi parametriksi lisätään float-muuttuja, joka on aikaisemmin Update-metodin sisällä määritelty. Tämän jälkeen halutaan saada animaatioiden vaihtuminen näyttämään sujuvammalta. Luodaan uusi muuttuja nimeltä locomationAnimationSmoothTime, jonka arvoksi asetetaan .1f. Muuttuja lisätään kolmanneksi parametriksi, jonka yhteydessä täytyy Time.deltaTime asettaa neljänneksi parametriksi (kuva 23).

```
// Update is called once per frame
void Update ()
{
    float speedPercent = agent.velocity.magnitude / agent.speed;
    anim.SetFloat("speedPercent", speedPercent, locomationAnimationSmoothTime, Time.deltaTime);
}
```

KUVA 23. CharachterAnimator skriptin Update-metodi

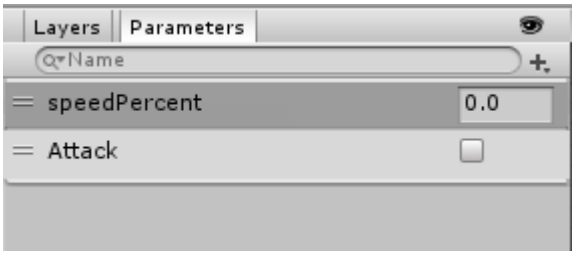
3.3.5 Pelihahmon Attack-animaation skriptin tekeminen

Pelihahmon liikuttamisen jälkeen pelihahmolle tehdään Attack-skripti, jolla pelihahmo voi tehdä vahinkoa vihollishahmoille. Tämän aloitetaan muokkaamalla animator-komponentin animator-ikkunaa. Aikaisemmin on animator-ikkunaa muokattu lisäämällä pelihahmolle animaatiot liikkumiselle. Nyt lisätään uusi animaatio, Attack-animaatio. Attack-animaatio lisätään siirtämällä animaatio animator-ikkunanäkymään. Siirtämisen jälkeen Attack-animaatiolle lisätään siirtymätilat edestakaisin Blend Treen kanssa (kuva 24).



KUVA 24. Attack-animaation siirtymätilat

Tämän jälkeen lisätään uusi parametri, jonka nimeksi annetaan Attack. Lisätty parametri on bool-tyyppinen (kuva 25).



KUVA 25. Parametri Attack

Seuraavaksi pelihahmolle lisätään uusi C#-skripti. Skriptin sisällä pelihahmon halutaan suorittavan Attack-animaation pelaajan klikatessa hiiren oikeanpuoleista näppäintä. Skriptin alussa pelihahmon Attack-animaation löytämiseksi täytyy viitata Animator-komponenttiin. Sen jälkeen tehdään uusi metodi nimeltä AttackKey. AttackKey-metodi halutaan toistaa Update-metodin sisällä. AttackKey-metodin sisälle kirjoitetaan jos-lause, jonka ehdoksi asetetaan hiiren oikeanpuoleisen näppäimen Input-arvo. Tämän lisäksi ehdoksi asetetaan pelihahmon manan arvo. Jokaisella kerralla, kun pelihahmolla tehdään Attack-animaatio, se menettää tietyn määrän manaa. Manan menettämiselle luodaan uusi int-muuttuja nimeltä ManaCostOfAttack, jonka arvoksi asetetaan 8. Manan sen hetkinen arvo haetaan PlayerGUI-skriptistä. Manan loppuessa pelihahmolla ei voida suorittaa Attack-animaatiota vaan joudutaan odottamaan manan kertymistä takaisin. Manan loppuminen ilmoitetaan Debug.Log-komennolla, joka lähettää viestin Unityn konsoliin (kuva 26).

```
void AttackKey()
{

    if(Input.GetMouseButtonDown(1) && playergui.CurrentMana >= ManaCostOfAttack )
    {

        animator.SetBool("Attack", true);

    }
    else if (Input.GetMouseButtonDown(1) && playergui.CurrentMana < ManaCostOfAttack)
    {
        animator.SetBool("Attack", false);
        Debug.Log("Wait for your mana!");

    }
```

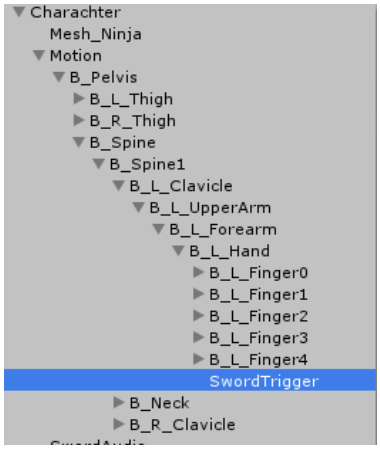
KUVA 26. Pelihahmon Attack-animaation skripti

3.3.6 Pelihahmon Damage-skriptin tekeminen

Pelihahmon Attack-animaation lisäämisen jälkeen täytyy tehdä uusi C#-skripti, Damage-skripti. Tämän aloitetaan lisäämällä uusi peliobjekti pelihahmo-objektin sisälle.

Monesti peliobjekteja ei ole tehty vain yhdestä kokonaisesta objektista, vaan ne sisältävät useita lapsiobjekteja. Näiden avulla peliobjektin sormet voidaan esimerkiksi eritellä kämmenestä tai jalat lantiosta. Tämän avulla voidaan hyödyntää pelihahmon sisältämää vasemman käden käsi-objektia. Vasemman käden käsi-objekti valitaan, koska pelihahmon

objektin sisällä ei ole eritelty erikseen pelihahmoon kiinnitettyä miekka-objektia. Vasemmankäden käsi-objektin alle halutaan luoda uusi lapsiobjekti. Uusi lapsiobjekti halutaan lisätä tähän, sillä Attack-animaation aikana objektin halutaan liikkuvan pelihahmon miekan mukaisesti. Objektiksi valitaan 3D-Capsule. Uudelle lapsiobjektille annetaan nimeksi SwordTrigger (kuva 27).



KUVA 27. SwordTrigger-objekti lisätty B_L_Hand-objektin lapsiobjektiksi

3D-Capsule-objektin kokoa ja muotoa muokataan niin, että saadaan upotettua se pelihahmon miekan sisälle (kuva 28). Kun objekti halutaan pelin aikana piilottaa, otetaan Mesh Render-komponentti pois käytöstä.

KUVA 28. 3D-Capsule objekti upotettuna pelihahmon miekan sisälle

Tämän jälkeen objektille lisätään Box-Collider. Collider-komponentti on lisätty Unityyn tarjoamaan törmäystunnisteen käyttämällä erilaisia ”rajauslaatikoita”. Tämän sisältä valitaan IsTrigger-ominaisuus. Kun IsTrigger-ominaisuus on valittu, colliderin törmätessä toisen colliderin kanssa se aiheuttaa OnTriggerEnter-eventin.

Pelihahmolla halutaan tehdä vahinkoa kuitenkin vain silloin, kun suoritetaan attack-animaatio. Attack-animaation aikana pitää kuitenkin collider-komponentin olla kosketuksissa Orc-objektin kanssa. Kun nämä molemmat ehdot täyttyvät OnTriggerEnter-eventin sisällä, vihollishahmo menettää elämäpisteitä annetun Damage-muuttujan määrän (kuva 29).

```
void OnTriggerEnter(Collider other)
{
    if (other.gameObject == Orc)
    {
        Hit = true;
        AnimatorStateInfo info = _animator.GetCurrentAnimatorStateInfo(0);

        if (info.fullPathHash == _playerAttackStateHash && Hit)
        {
            Debug.Log("You did damage to enemy!");

            enemyHealthGUI.TakeDamage(Damage);
        }
    }
}
```

KUVA 29. OnTriggerEnter-metodi

TakeDamage-metodi määritellään EnemyHealthGUI-skriptin sisällä (kuva 30).

```
public void TakeDamage(int amount)
{
    CurrentHealth -= amount;
}
```

KUVA 30. TakeDamage-metodi

3.3.7 Ääniefektien lisääminen

Lopuksi pelihahmolle lisätään ääniefekti. Ensimmäinen ääniefekti halutaan lisätä Attack-animaation aikana. Ääniefektit voi ladata Asset Storesta, jonka jälkeen voidaan aloittaa niiden lisääminen.

Ensimmäiseksi pelihahmo-objektin sisälle luodaan uusi peliobjekti. Objektin nimeksi annetaan SwordAudio. Tämän jälkeen objektille lisätään äänilähde eli Audio Source -komponentti. Seuraavaksi lisätään pelihahmolle uusi C#-skripti. Skriptin alussa viitataan AudioSource sekä AudioClip-komponentteihin ja asetetaan ne julkisiksi. Tämän jälkeen li-

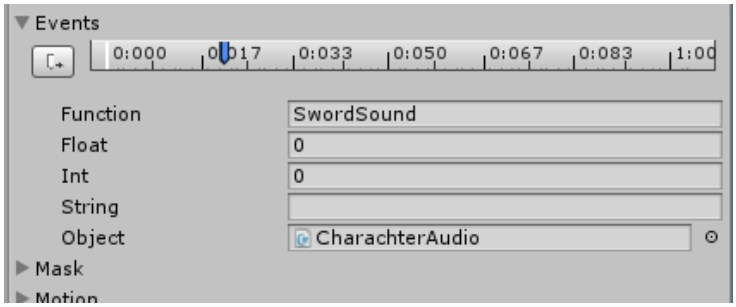
sätään SwordSound-metodi. Metodin sisällä toistetaan äänileike (kuva 31). Tämän jälkeen voidaan äänilähteeksi siirtää SwordAudio-objekti sekä valita ladatuista ääniefekteistä haluttu äänileike.

```
public class CharachterAudio : MonoBehaviour
{

    public AudioClip swordSound;
    public AudioSource AudioS;

    public void SwordSound()
    {

        AudioS.PlayOneShot(swordSound);

    }
}
```

KUVA 31. CharachterAudio-skripti

Kun ääniefekti halutaan saada toistettua Attack-animaation aikana, täytyy muokata Attack-animaatiota. Ensiksi avataan Attack-animaation muokkaus-tila, jossa päästään lisäämään animaatiolle tapahtumia. Animaatiotapahtumia saadaan lisättyä avaamalla tapahtuma-tila asetukset.

Tämän sisällä voidaan tietylle animaation aikajanalle asettaa uusi tapahtuma. Kun valittu saadaan selville, lisätään objektin kohtaan ääniefektille tehty skripti. Funktion kohdalle kirjoitetaan skriptin sisällä olevan funktion nimi, SwordSound (kuva 32).



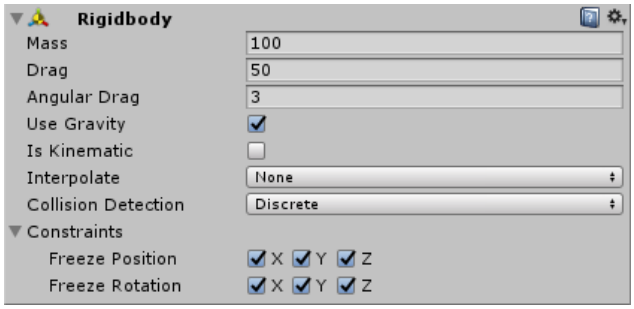
KUVA 32. Animaatiotapahtuman lisääminen

3.4 Vihollishahmo

Kun pelihahmo oli saatu valmiiksi, voitiin aloittaa vihollishahmon tekeminen. Tämä aloitetaan lataamalla Orc-paketti käyttäen Asset Storea. Paketin lataamisen jälkeen sen sisältämä Orc-elementti siirretään pelikenttään (kuva 33).

KUVA 33. Orc-objektin lisääminen

Seuraavaksi lisätään Rigidbody-komponentti. Rigidbody-komponentin avulla objektille voidaan kohdistaa voimaa sekä fysikaalisia ilmiöitä. Asetuksista asetetaan objektille painovoima. Tämän jälkeen pelihahmon ei haluta pystyvän puskemaan objektia. Sen takia objektin liike rajataan ”jäädyttämällä” sen sijainti sekä pyöriminen kaikilla akseleilla (kuva 34).

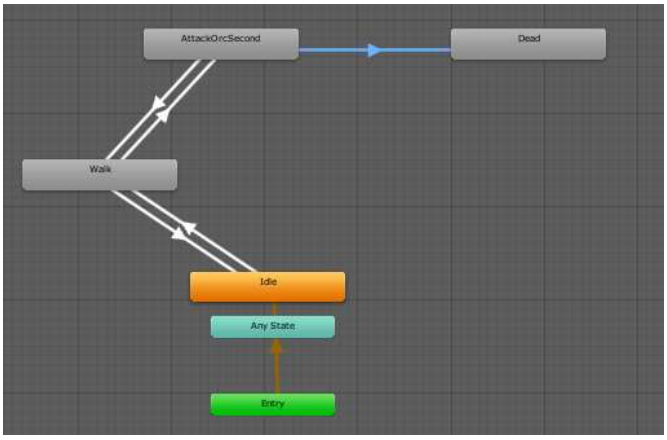


KUVA 34. Rigidbody-asetukset

Rigidbody-komponentti ei kuitenkaan reagoi törmäyksiin ilman collider-komponenttia. Sen takia lisätään Box Collider-komponentti. Collider-komponenttia hyödynnetään, kun halutaan tunnistaa pelihahmon miekka-objektin collider-komponentin osuminen.

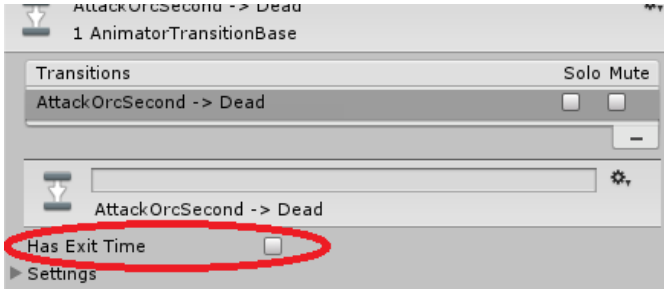
3.4.1 Animaatioiden lisääminen

Seuraavaksi aloitetaan animaatioiden lisääminen. Orc-paketin lataamisen mukana tuli useita animaatiota, joita pystytään hyödyntämään. Näistä valitaan kävely-, lyönti-, idle- sekä kuolema-animaatio. Animator-komponentin lisäämisen jälkeen lisätään animator-ikkunan sisälle tarvittavat animaatiot. Animaatioiden välille lisätään siirtymätilat ja idle-animaatio asetetaan oletustilaksi (Kuva 35).



Kuva 35. Animator-ikkuna näkymä Orc-objektin animaatiotiloista

Tämän jälkeen lisätään parametrit. Parametreiksi asetetaan kolme bool-muuttuja sekä yksi trigger. Kun animaatiotilasta halutaan siirtyä toiseen, pitää skriptissä bool-muuttuja asettaa true-tilaan. Vihollishahmon elämäpisteiden loputtua asetetaan trigger-parametri true-tilaan, jolloin suoritetaan kuolema-animaatio. Animaatioiden siirtymätiloista poistetaan myös animaation suorittaminen loppuun, ennen kuin se siirtyy toiseen animaatioon. Tämä saadaan poistamalla Has Exit Time -ominaisuus animaation siirtymätilan inspector-näkymästä (kuva 36).



KUVA 36. Inspector-näkymä animaatioiden välisestä siirtymätilasta

3.4.2 OrcChase-skriptin tekeminen

Yksi projektin tavoitteista on saada vihollishahmo seuraamaan pelihahmoa sen saavuttua tietylle etäisyydelle. Tämä aloitetaan lisäämällä vihollishahmolle NavMesh Agent -komponentti. NavMesh Agent -komponentin avulla asetetaan vihollishahmolle nopeus, kiihtyvyys, kulmanopeus sekä pysähtymisetäisyys.

Tämän jälkeen luodaan uusi C#-skripti. Ensimmäiseksi lisätään uusi float-muuttuja nimeltä lookRadius. Tätä pidetään säteenä, jonka sisälle pelihahmon tullessa vihollishahmo aloittaisi seuraamisen. Tämän jälkeen Update-funktion sisällä selvitetään etäisyys pelihahmon sekä vihollishahmon välillä. Jos etäisyys on pienempi kuin lookRadius-arvo, vihollishahmo aloittaa kävely-animaation. Kävely-animaation yhteydessä aktivoidaan NavMesh Agent -komponentti. Agentin päämääräksi asetetaan pelihahmon kohde. Kun etäisyys kasvaa liian suureksi, vihollishahmo aloittaa idle-animaation sekä NavMesh Agent -komponentti poistetaan käytöstä (kuva 37).

```
if(distance <= lookRadius)
{
    animations.SetBool("isWalking", true);
    animations.SetBool("isIdle", false);
    agent.GetComponent<NavMeshAgent>().enabled = true;
    agent.SetDestination(target.position);
}
else if(distance >= lookRadius)
{
    animations.SetBool("isWalking", false);
    animations.SetBool("isIdle", true);
    agent.GetComponent<NavMeshAgent>().enabled = false;
}
```

KUVA 37. Ehdot animaatioiden aloittamiselle

Tämän jälkeen halutaan vihollishahmon pysähtyvän tietylle etäisyydelle sen saavuttua kohteeseen. Tähän käytettiin NavMesh Agent -komponentin pysähtymisetäisyys-ominaisuutta hyväksi. Tämän avulla verrataan etäisyyden arvoa pysähtymisetäisyyden arvon kanssa. Jos etäisyys on pienempi tai yhtä suuri kuin agentin pysähtymisetäisyys, lopetetaan kävely-animaatio ja aloitetaan hyökkäys-animaatio (kuva 38).

```
if(distance <= agent.stoppingDistance )
{
    animations.SetBool("isWalking", false);
    animations.SetBool("isAttacking", true);

    if (enemyHealth.CurrentHealth > 0)
    {
        FaceTarget();
    }
}
else
{
    animations.SetBool("isAttacking", false);
}
```

KUVA 38. Lyönti-animaation aloittaminen pysähtymisetäisyydellä

Tämän jälkeen lisätään vielä FaceTarget-funktio, jonka avulla saadaan vihollishahmo kääntymään aina kohti pelihahmoa. Funktio suoritetaan vain silloin, kun vihollishahmo saavuttaa pysähtymisetäisyyden (kuva 39).

```
void FaceTarget()
{
    Vector3 direction = (target.position - transform.position).normalized;
    Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0, direction.z));
    transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime * 5f);
}
```

KUVA 39. FaceTarget-metodi

3.4.3 Vihollishahmon viimeisteleminen ja ääniefektien lisääminen

Kun vihollishahmo saadaan animoitua sekä seuraamaan pelihahmoa sen tultua tarpeeksi lähelle, lisätään sille myös elämäpisteet. Elämäpisteitä varten lisätään uusi C#-skripti nimeltä EnemyHealthGUI. Aikaisemmin pelihahmolle on tehty elämäpisteet, joten tätä hyödyntäen saadaan myös vihollishahmolle tehtyä GUI-näkymä. Vihollishahmon elämäpisteet halutaan kuitenkin nähdä vasta, kun se aloittaa seuraamisen. Etäisyyttä verrataan Chase-skriptin lookRadius-muuttujan kanssa. Kun etäisyyden arvo on pienempi kuin lookRadius arvo, GUI-näkymä näytetään. Muuten se pidetään false-tilassa (kuva 40).

KUVA 40. Vihollishahmon elämäpisteiden näyttäminen ja piilottaminen

Vihollishahmon elämäpisteiden loputtua, vihollishahmo suorittaa kuolema-animaation. Kuolema-animaation jälkeen Orc-objekti upotetaan tietyllä nopeudella maan läpi (kuva 41).

```
if (sinking)
{
    transform.Translate(-Vector3.up * speed * Time.deltaTime);
}
```

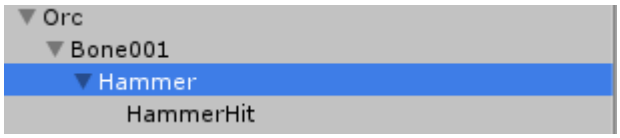
KUVA 41. Objektin upottaminen

Tämän yhteydessä Orc-objektilta poistetaan NavMesh Agent -komponentti sekä Rigid-body-komponentti ja uppoamisen jälkeen Orc-objekti tuhotaan (kuva 42).

```
if(CurrentHealth <= 0)
{
    animis.SetTrigger("isDead");
    orcChase.GetComponent<OrcChase>().enabled = false;
    agent.GetComponent<NavMeshAgent>().enabled = false;
    sinking = true;
    Destroy(orc, destroyTime);

}
```

KUVA 42. Orc-objektin elämäpisteiden loppuminen

Tämän jälkeen Orc-objektin halutaan myös tekevän vahinkoa pelihahmolle. Se tehdään samalla tavalla kuin pelihahmon miekka-objektin kanssa aikaisemmin. Lisätään Hammer-objektin alle uusi lapsiobjekti nimeltä HammerHit (kuva 43).



KUVA 43. Lapsiobjektin lisääminen

HammerHit-objektille lisätään Box-collider ja valitaan IsTrigger-ominaisuus. Tämän jälkeen hyödynnetään aikaisempaa Damage-skriptiä, joka on tehty pelihahmolle. Skriptin sisällä haetaan kuitenkin Orc-objektin lyönti-animaatio. Animaation aikana sen osuessa pelihahmoon pelihahmo menettää tietyn määrän elämäpisteitä (kuva 44).

```
void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Player"))
    {
        playerInRange = true;
        AnimatorStateInfo info = anim.GetCurrentAnimatorStateInfo(0);

        if (info.fullPathHash == _enemyAttackStateHash && playerInRange == true)
        {
                Debug.Log("Enemy hits you!");
                playerGUI.TakeDamage(attackDamage);
        }
    }
}
```

KUVA 44. OnTriggerEnter-metodi

Lopuksi vihollishahmolle lisätään ääniefektit. Tarvittavat ääniefektit voi ladata Asset Storesta. Ääniefektien lisäämisessä käytetään samaa tekniikkaa kuin pelihahmon ääniefektien kanssa. Lisätään Orc-objektille kaksi uutta peliobjektia. Nimeämisen jälkeen molempiin lisätään AudioSource eli äänilähde-komponentti.

Seuraavaksi lisätään uusi C#-skripti. Skriptissä funktioiden sisällä toistetaan äänileike (kuva 45). Äänileike toistetaan animaatioiden aikajanalle määrätyssä kohdassa kutsumalla funktiota.

```
public void MonsterAttack()
{

    audioSo.PlayOneShot(monsterBite, 0.3f);
}

void FootStep()
{
    AudioS.PlayOneShot(footStep, 1.4f);
}
```

KUVA 45. Funktioiden sisällä toistetaan äänileike

Tämän jälkeen voidaan pelin sisällä pelihahmolla taistella vihollishahmon kanssa (kuva 46).

KUVA 46. RPG-peli pelitilassa

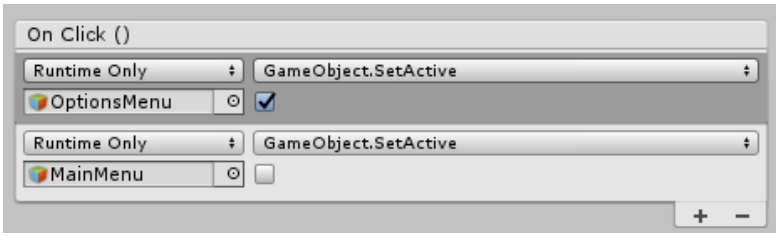
3.5 Päävalikko

Viimeiseksi tehtiin peliin päävalikko. Päävalikon tekemiseen käytettiin apuna opetusvideota Youtubesta (10). Päävalikon tekeminen aloitetaan lisäämällä uusi Scene eli näkymä. Uusi näkymä saadaan lisättyä valitsemalla **File -> New Scene**. Uudelle näkymälle lisätään UI-elementeistä Canvas-objektin. Sen jälkeen lisätään Image-elementti, joka asetetaan yhtä leveäksi ja korkeaksi kuin Canvas-elementti. Tämän jälkeen lisätään kaksi tyhjää lapsiobjektia, jotka nimetään MainMenuksi sekä OptionsMenuksi (kuva 47).

KUVA 47. Canvas-objekti

MainMenu-objektin sisälle lisätään kolme Button-elementtiä. Options-buttonia klikatessa halutaan siirtyä OptionsMenu-valikkoon. Tämän tekemiseen täytyy muokata Options-but-

tonin OnClick-funktiota. Kun Options-buttonia klikataan, halutaan saada näkyviin vain OptionsMenu-valikko. Se tehdään mahdolliseksi asettamalla funktio GameObject.SetActive true-tilaan OptionsMenu-valikon kohdalla. Tämän yhteydessä MainMenu-valikon GameObject.SceneActive asetetaan false-tilaan. OptionsMenu-valikko ei sisällä muuta vaihtoehtoa kuin Back-buttonin. Tämän avulla OptionsMenu-valikko piilotetaan ja MainMenu-valikko asetetaan näkyviin (kuva 48). Projektin jatkoa varten se on kuitenkin hyvä lisätä.



KUVA 48. OptionsMenu valikon asettaminen näkyviin

Seuraavaksi täytyy saada oikeat toiminnot Play ja Exit -buttonille. Aluksi luodaan uusi skripti nimeltä MainMenu. MainMenu-skriptin sisälle luodaan kaksi funktiota, PlayGame sekä QuitGame. Tämän jälkeen Play-buttonin OnClick-funktion halutaan suorittavan MainMenu-skriptin PlayGame-funktio, minkä jälkeen scenen vaihtuminen on mahdollista (kuva 49).

```
public void PlayGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```

KUVA 49. PlayGame-funktio MainMenu skriptissä

Sen jälkeen kirjoitetaan QuitGame-funktio. Quit-funktion avulla peli sammutettiin (kuva 50).

```
public void QuitGame()
{
    Debug.Log("Quit");
    UnityEditor.EditorApplication.isPlaying = false;
    Application.Quit();
}
```

KUVA 50. QuitGame-funktio MainMenu skriptissä

Tämän jälkeen ladataan Asset Storesta ilmainen TextMeshPro-paketti. Paketin avulla saadaan muokattua tekstin tyyliä ja fonttia. Tämän jälkeen päävalikkonäkymä saatiin tehtyä valmiiksi (kuva 51).

KUVA 51. Päävalikko

4 YHTEENVETO

Opinnäytetyön tavoitteena oli kehittää yksinkertainen ja toimiva RPG-pelin prototyyppi käyttäen Unity3D-pelimoottoria sekä saada lisää kokemusta pelinkehitykseen. Kaikkiin näihin tavoitteisiin päästiin, vaikka projektin alussa jouduttiinkin moniin ongelmiin. Projektin sisällön rajaaminen tuotti ongelmia, sillä RPG-pelien sisältö on laaja ja kaiken tämän luominen vaatisi isomman ryhmän tekemään projektia. Sisällön rajaamisen jälkeen päästiin kuitenkin aloittamaan projekti ja siitä eteenpäin saatiin lisättyä tarvittavat toiminnallisuudet peliin. C#-ohjelmointikieli oli osaksi tuttu, mutta projektin yhteydessä sen osaaminen lisääntyi huimasti.

Opin projektin aikana karsimaan tiettyjä alueita, koska tietyn toiminnallisuuden tekeminen täysin samanlaiseksi kuin nykyajan pelattaviin RPG-peleihin ei vain ollut omien taitojeni avulla mahdollista. Ymmärsin, että vaikka pelin visuaalinen puoli ei välttämättä näyttänyt hyvältä, pelin toimivuuden kannalta se ei haitannut. Vaikka olinkin aikaisemmin käyttänyt Unity3D:tä, opin projektin aikana paljon uutta.

Projektin aikana syntynyt peli ei tullut valmiiksi. Peliin saatiin kuitenkin luotua hyvä alku, josta on helppo tulevaisuudessa jatkaa eteenpäin. Työtä tehdessäni kiinnostukseni pelien kehittämiseen kasvoi vain enemmän. Toiveenani olisi myös, että voisin joskus olla mukana isommassa projektissa, jossa pääsisin isomman ryhmän kanssa kehittelemään toimivan RPG-pelin.

LÄHTEET

1. Unity (pelimoottori). 2017. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Unity_(pelimoottori). Hakupäivä 6.5.2018.

2. Tietokoneroolipeli. 2017. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Tietokoneroolipeli. Hakupäivä 6.5.2018.

3. Panu Roivas. PC-roolipelien historia. 2012. 30+ vuotta PC-roolipelejä. Saatavissa: https://www.hardware.fi/artikkelit/artikkeli.cfm/kuvissa_pc-roolipelien_historia. Hakupäivä 6.5.2018.

4. Immersio. 2017. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Immersio. Hakupäivä 22.5.2018.

5. Diablo I. 2018. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Diablo_(video_game). Hakupäivä 14.5.2018.

6. Diablo II. 2018. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Diablo_II. Hakupäivä 6.5.2018.

7. Diablo III. 2017. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Diablo_III. Hakupäivä 6.5.2018.

8. The Elder Scrolls V: Skyrim. 2018. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/The_Elder_Scrolls_V:_Skyrim. Hakupäivä 6.5.2018.

9. Massiivinen monen pelaajan verkkoroolipeli. 2017. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Massiivinen_monen_pelaajan_verkkoroolipeli. Hakupäivä 14.5.2018.

10. Start Menu in Unity. 29.11.2017. Saatavissa: https://www.youtube.com/watch?v=zc8ac_qUXQY&t. Hakupäivä 14.5.2018.