3D videopelien tekoäly, grafiikka ja pelimoottorit

Oulun yliopisto
Tieto- ja sähkötekniikan tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
LuK-tutkielma
Paula Häyrynen
14.3.2021

Tiivistelmä

Pelaaminen on ollut kasvava harrastus ja pelejä julkaistaan nykyaikana todella paljon. Tietokoneella pelattavat pelit ovat kolmanneksi suosituimpia pelejä konsoli- ja mobiilipelien jälkeen. Myös pelien lukumäärä on kasvanut, mikä johtuu osittain siitä, että niiden tekemiseen on olemassa erilaisia ohjelmistoja. Lisäksi pelit ovat kehittyneet ja 2D pelien lisäksi on olemassa 3D pelejä.

Tässä kirjallisuuskatselmuksessa käydään lävitse videopelien kannalta 3 oleellista asiaa; pelien tekoäly, 3D objektien luonti ja pelimoottorit.

Pelien tekoäly voi olla kovakoodatut scriptit, joissa ennaltamääritellyissä tilanteissa tekoäly toimii tietyllä tavalla. Peleissä käytetään myös paljon agentteja, finite state machinea ja uudeksi trendiksi on noussut behaviour trees. Erilaiset tekoälyt sopivat erilaisiin peleihin, joten niistä on vaikea valita vain yhtä parasta.

3D objekteja luoomiseen on olemassa erilaisia tapoja, kuten mallintaminen (low poly tai high poly) ja veistäminen. Mallintamisen jälkeen objekti mapataan, lisätään tekstuurit, muovataan ja animoidaan erilaisiin tilanteisiin sopivaksi. Objektien luontiin on olemassa erilaisia ohjelmia, kuten myös tekstuurien tekemiseen.

Myös pelimoottoreita on olemassa erilaisia, mutta yleistä näillä kaikilla on reaaliaikainen renderointi, törmäyksen havaitseminen, fyysisyys objekteissa, tekoäly, skriptien teko mahdollisuus ja editori.

Avainsanat
Tekoäly, 3D mallinnus, pelimoottori

Valvoja
Ph.D Mikko Rajanen

Sisällysluettelo

1. Johdanto

Pelit ovat olleet kovassa nousussa jo useiden vuosien ajan. PC pelit ovat kolmanneksi suosituin pelimuoto mobiilipelien ja konsolipelien jälkeen (Newzoo, 2018). Sijastaan huolimatta juuri tietokoneille ja Maceille luodaan eniten pelejä (53 %) pelialalla (WePC, 2019).

Erään tutkimuksen mukaan pelin tuottamiseen meni 50 % työajasta, koska peliin tuli luoda grafiikat ja ohjelmoida se. Ennakkotuotanto ja pelin testaaminen veivät molemmat 20 % työajasta ja jälkituotanto vei 10% (Torres-Ferreyros, Festini-Wendorff & Shiguihara-Juarez, 2017).

Pelien tuottamisessa tärkeässä roolissa on ohjelmointi, etenkin toimivan tekoälyn luonti ei pelattavilla hahmoilla. Tekoälyllä luodut vastustajat eivät saa olla liian helppoja tai liian vaikeita pelaajalle, jotta pelaajan mielenkiinto peliin säilyy. Pelipuolella tekoäly on erilaista kuin esimerkiksi IoT -laiteissa. Peleissä tekoäly ei jatkuvasti opi uutta tai se on voitu luoda kovakoodaamalla. Tekoäly on tärkeä osa pelejä, koska sillä saadaan pelimaailmaan luotua vastustajia tai kumppaneita, jotka toimivat itsenäisesti ja reagoivat pelaajan toimiin tarvittaessa.

Peligrafiikka ei pelaamisen kannalta ole ehkä yksi olennaisimmista osista ja yksinkertaisellakin grafiikalla peli voi menestyä. Silti enenemissä määrin luodaan 3D grafiikalla tehtyjä pelejä. Pelien graafikot panostavat enemmän ja enemmän realistiseen lopputuloksen saamiseksi, kuitenkin huomioiden mahdolliset rajoitukset, jottei 3D objektit ole liian raskaita pyörittää peleissä (Wallin. 2015. s. 7).

Pelimoottorit ovat helpottaneet pelien luomista. Niiden avulla voidaan luoda erilaisia pelimaailmoja ja -kenttiä helposti. Aiemmin pelimoottorit saattoivat olla vain pelifirman omia sisäisiä työkaluja, mutta nykyään niitä saa myös yksittäiset henkilöt. Tunnetuimmat markkinoilla olevat pelimoottorit ovat Unity3D ja Unreal Engine. Näiden lisäksi on pienempiä tai OSS pelimoottoreita.

Tässä kirjallisuuskatsauksessa on tarkoitus tarkastella pelien tekoälyä, 3D objektien luomista peleihin ja tutustua pelimoottoreihin, koska ne ovat olennaisia osia videopelien luomisessa.

Tutkimuskysymykset:
- Mitä eri tekoälyjä on peleille olemassa?
- Mitä eri työkaluja on olemassa 3D objektien luomiseen?
- Kuinka luodaan 3D objekteja?
- Millaisia pelimoottorit ovat ja miten ne auttavat pelien luomisessa?

2. Tekoäly

Tässä kappaleessa käymme läpi mitä tekoäly on peleissä ja muutamia yleisimpiä tekniikoita, joilla peleille on luotu tekoälyä. Vaikka tekoäly on yksi tärkeä osa peliä ja tekee pelistä mielenkiintoisen (Turan & Çetin. 2019), se yleensä implementoidaan peliin vasta viimeisten joukossa (Sweetser & Wiles, 2002)

2.1 Yleistä

Kaikessa yksinkertaisuudessaan tekoälyn avulla tietokone kykenee loogiseen ajattelutapaan, johon yleensä ovat olleet kykeneviä ihmiset ja eläimet. Pelien kehittämisessä tekoälyn algoritmeissa keskitytäänkin rakentamaan algoritmeja, jotka saavat tekoälyn pelihahmon käyttäytymään ihmismäisesti tai eläimellisesti (Millington & Funge, 2009, s. 3–6). Tekoälyn lisääminen tekee peliympäristöstä interaktiivisemman ja uskottavamman, etenkin realistisissa peleissä ja virtuaalisissa ympäristöissä. (Turan & Çetin. 2019). Videopelien tekoälyn luominen on haastavampaa kuin esimerkiksi lautapelien tekoälyn luominen (Robertson & Watson, 2014).

Millington & Funge (2009, s. 8–9) esittävät kirjassaan yhden yleisen mallin, jonka avulla saadaan aikaiseksi älykkäitä pelihahmoja. Mallissa yksittäinen tekoäly ja tekoäly ryhmä saa aluksi informaatiota maailmasta, jossa ne ovat. Malli on jaettu tekoälyn tehtävä kolmeen osioon, joista ensimmäisessä tekoäly liikkuu, päättää mitä tekee ja strategiaan. Liikehdintä ja päätöksen tekeminen kuuluvat yksittäisille hahmoille ja strategian tekeminen kuuluu tekoäly ryhmälle. Näiden päätösten jälkeen toiminnot näkyvät peliruudulla. Tämä malli pohjautuu agentti pohjaiseen malliin, jossa itsehallinnollinen pelihahmo saa informaatiota pelin datasta, joiden pohjalta tekee toimintoja. Aluksi siis tarkastellaan sitä, miten yksittäiset hahmot käyttäytyvät ja tekoäly tukee niiden implementointia. Tämä on niin sanottu “bottom-up” -tapa. Ei-agentti tavassa taasen mennään päinvastoin ylhäältä alaspäin ja rakennetaan yksittäisiä systeemejä simuloimaan kaikkea. Tekoälyn halutaan käyttäytyvän mahdollisimman uskottavasti, koska huonosti toteutettu tekoäly vaikuttaa negatiivisesti pelikokemukseen. Peleissä käytettäviä tekoälyjä on olemassa monenlaisia juuri sen takia, että niillä yritetään saada mahdollisimman realistinen tekoäly aikaiseksi peliin (Turan & Çetin. 2019).

2.2 Eri teknologiat

On olemassa monia erilaisia teknologioita, joilla luodaan peleihin tekoälyä, kuten esimerkiksi Finite State Machinesit, Neural Networksit (Sweetser & Wiles, 2002, Lu & all. 2013), Behavior trees (Agis, Gottifredi & García, 2020), skriptaus, agentit, Flocking, Fuzzy Logic. Jokaisella teknologialla on omat vahvuutensa ja heikkoutensa, mutta niistä huolimatta suosituimpia teknologioita ovat olleet Finite state machine, skriptaus, agentit ja flocking (Sweetser & Wiles, 2002). Toisaalta taasen, pelit ja niihin liittyvä muu teknologia on kehittynyt paljon, jota myös Sweetser & Wiles (2002) ennustivat ja jo silloin he miettivät, että peleihin tullaan haluamaan parempia tekoälyjä, joita Fuzzy Logic, Neural Networksi ja Genetic Algorithms voisivat tarjota.

2.2.1 Agentit

Peleissä, joissa on tekoäly, käytetään agenttia jossain muodossa. Pelit ovat ihanteellinen ympäristö agenteille, koska niissä ympäristöstä saatava tieto on rajallista ja agentin tulee tehdä päätökset aika- ja painerajotteiden sisällä. Agentit voidaan luoda esimerkiksi eri teknologioiden avulla, joita on lueteltu aiemmassa kappaleessa tai näiden kombinaatioilla. Agenttia luodessa on tärkeää miettiä tarkkaan agentin arkkitehtuuri. Agentti, joka on hyvin jäsennelty ja jaoteltu kommunikoiviin tasoihin / moduuleihin, joilla on omat vastuualueensa, on huomattavasti parempi peleissä kuin sellaiset agentit, joissa on vain pari valtavaa State Machinesia. Muutoinkin tällaisten isojen State Machinesien kehittäminen voi käydä hallitsemattomaksi kehityksen aikana. Toinen tärkeä asia on se, onko agentti voittotavoitteinen, reagoiva vai näiden kahden sekoitus. Reagoiva agentti sopii esimerkiksi first-person ammuntapeliin tai roolipelaamiseen, koska sen tarvitsee reagoida siihen, mitä pelissä tapahtuu juuri nyt. Voittotavoitteinen agentti sopii taasen paremmin strategiapeleihin, koska niissä pitää tarkasti miettiä omia siirtoja ja pitää tietää mitä aiemmin on jo tapahtunut pelissä (Sweetser & Wiles, 2002).

2.2.2 Finite State Machine

Finite State Machine (FSM) on ollut yksi suosituimmista tekniikoista, joilla on luotu peleihin tekoälyä. FSMa voidaan käyttää pelissä lukemattomin eri tavoin, kuten pelimaailman hoitamisessa, pelin tai peli objektin tilan hoitamiseen, simuloimaan ei-pelattavan hahmon tunteita tai laittaa jäsentämään pelaajan antamia syötteitä. Sitä on käytetty esimerkiksi Doom, Age of Empires, Half-Life ja Quake -peleissä (Sweetser & Wiles, 2002). FSMssa on tiettyjä tiloja, jotka vaihtuvat sääntöjen takia tai pelin edetessä (Turan & Çetin. 2019). Suosiota selittää se, että kyseinen tekniikka on helppo ymmärtää ja ohjelmoida. Lisäksi FSMlla on yksinkertaisimmat laskennalliset ohjelmat, alhainen laskennallinen yläraja, sitä voidaan käyttää yhdessä muiden tekniikoiden kanssa ja FSM antaa paljon voimaa, verrattuna sen monimutkaisuuteen (Sweetser & Wiles, 2002). Toisaalta taasen, jos peliä ei suunnittele ja rakenna hyvin, FSMsta voi tulla vaikeasti ylläpidettävä ja sitä voi olla hankala parantaa (Sweetser & Wiles. 2002 & Turan & Çetin. 2019). Muita huonoja puolia ovat skaalautuvuuden heikkous ja se, että ohjelmoija joutuu ottamaan huomioon kaikki mahdolliset skenaariot ja tilanteet, joita pelin tekoäly voi kohdata (Sweetser & Wiles, 2002). Joidenkin mielestä FSM on hieman vanhanaikainen verrattuna muihin tekniikoihin. Toisaalta kehittyneempiä tekniikoita ei voi välttämättä käyttää joissain peleissä erinäisten syiden takia, jolloin FSM on hyvä vaihtoehto tekoälyn luomiselle (Sweetser & Wiles, 2002).

2.2.3 Neural Network

Neural Network on helpompi koodata kuin erilaiset State Machinesit (Turan & Çetin. 2019) ja ne ovat joustavia, ei-lineaarisia ja ei-deterministisiä (Sweetser & Wiles, 2002). Lisäksi ANNin etuna on sen kestävyys korruptoituneen syötteen suhteen ja työskentelynopeus (Turan & Çetin. 2019). Tosin Neural Networksit tarvitsevat virittämistä, ne ovat monimutkaisia, muuttujien valinta on hankalaa ja ne käyttävät intensiivisesti resursseja. Turan & Çetin (2019) esittävät artikkelissaan erilaisia Neural Networkkeja, joita on käytetty erilaisiin tilanteisiin. Näitä ovat Artificial Neural Network (ANN), Back Propagation Neural Network (BP) ja Enforce Sub-Population Neural Network (ESP). ANN perustuu yksinkertaistettuun malliin aivoista, ja se voidaan myös opettaa matkimaan oikeaa pelaajaa pelissä. Tieto ensin kerätään ympäristöstä, jonka jälkeen se käy oppimisprosessin läpi ja viimeisenä kerätty tieto varastoidaan. Se voidaan antaa tehdä itse päätös tai antaa tulkita dataa, joka perustuu aikaisemmin annetun

syötteeseen ja ulostuloon. ANN valitsee sellaisen toiminnon, joka on suoritettu aiemmin jossain samankaltaisessa tilassa. ANNia luodessa on hyvä pitää syötteiden määrä pienenä ja ottaa mukaan alussa vain olennaiset muuttujat ja vasta myöhemmin lisätä muuttujia tarpeen mukaan. Oikeiden syötteiden määrittäminen tosin voi olla haastavaa, koska pelimaailmasta pystyy saamaan paljon informaatiota ja jos tulee valitsemaan sellaiset muuttujat, joiden syötteet antavat heikosti tietoa pelimaailmasta, lopputuloksena on epäonnistunut tekoäly. Yleensä kun ANNia ohjelmoidaan peliin, sitä koulutetaan ohjelmoinnin aikana ja kun koulutus on saatu päätökseen, ohjelma asetukset lukitaan ennen kuin peli julkaistaan. Tämä siksi, ettei tekoäly oppisi huonoja asioita. ANNia on käytetty Black & White, BC3K, Creatures ja Heavy Gear -peleissä ja jo vuonna 2002 ennustettiin, että kyseistä tekoälyä tultaisiin käyttämään yhä enemmän maksullisissa peleissä (Sweetser & Wiles, 2002).

2.2.4 Fuzzy Logic

Fuzzy Logic on hyvin tunnettu tekniikka ja siinä on useita etuja, verrattuna muihin tekniikoihin, kuten alhaiset laskennalliset kustannukset ja sen käytös muistuttaa paljon ihmisen käyttäytymistä, mutta sitä voidaan käyttää myös esimerkiksi eläimien tekoälynä (Turan & Çetin. 2019). FL on samankaltainen booleanin logiikan kanssa, paitsi että FLssa on laajemmin eri vaihtoehtoja, joista valita tilanteeseen sopiva, kun taas booleanissa vaihtoehtoja on kaksi (tosi / epätosi). Toinen eroavaisuus on siinä, että FLa voi tehdä päätöksen, vaikka data ei olisi täydellistä ja se sisältäisi virheitä. FLn etuna on se, että se voi esittää konspeti käyttämällä pienen määrän sameita arvoja. Sitä ei tarvitse kovakoodata (Sweetser & Wiles, 2002) tai luoda monimutkaisia matemaattisia malleja (Turan & Çetin. 2019). FLia voidaan käyttää pelien tekoälyssä päätöksenteossa ja tekoälyn käyttäytymiseen tiettyyn tilanteeseen ja niitä voidaan käyttää myös silloin, jos tekoälyn tekemiseen ei riitä yksinkertainen ratkaisu, ongelma ei ole lineaarinen, vaan tekoälyyn tarvitaan asiantuntemusta ja enemmän joustavuutta sekä muuttujia (Sweetser & Wiles, 2002). FLn säännöt ovat erillisiä toisistaan, jonka takia näitä sääntöjä voidaan lisätä tai poistaa helposti logiikasta, mikä tekee FLin päivittämisestä helpompaa (Turan & Çetin. 2019). Kuitenkin FL on monimutkainen rakentaa (Sweetser & Wiles, 2002), koska tekijän täytyy tunnistaa tarpeelliset sääntöjoukot sekä syötteiden ja tulostusten muuttujat (Turan & Çetin. 2019). FLia on käytetty esimerkiksi suositussa The Sims pelissä, sekä SWAT 2, Call of Power, Close Combat ja Petz -peleissä (Sweetser & Wiles, 2002).

2.2.5 Behavior Trees

Behavior Trees (BT) ovat todella suosittuja nykyään videopelien ei-pelattavien hahmojen tekoälyssä, koska niillä pystytään luomaan monimutkaista käytöstä tekoälylle ohjelmoimalla ensin toimintoja ja vasta sitten tarvitsee suunnitella puun rakenne. BTsa lehtien solmut ovat toimintoja ja niiden sisällä olevat solmut määrittelevät päätöksentekoa. Ne ovat myös helppo suunnitella, testata ja debugata. Ja esimerkiksi FSMen verrattuna BT on modulaarisempi, skaalautuvaisempi ja luotettavampi. BTta on kehitetty myös astetta parempi versio event-driven behavior trees (EDBT), jossa puu pystyy käsittelemään suoritustaan ottamalla käyttöön uuden noden, joka voi vaikuttaa tapahtumaan ja keskeyttää ne nodet, jotka sillä hetkellä ovat pyörimässä. EDBTta saatetaan kuitenkin kutsua BT nimellä, koska se on helpompi muistaa. BTta on käytetty suosituissa Halo 2 ja Halo 3 -peleissä (Agis, Gottifredi & García, 2020).

2.2.6 Skriptaus

Skriptausta käytetään paljon peleissä ja sitä voidaan hyödyntää myös tekoälyssä. Se on eräänlainen ohjelmointikieli, jonka avulla saadaan tehtyä monimutkaiset asiat helpommin, kuten vaikka hahmon liikkuminen pelissä. Sen avulla voidaan vaikuttaa peliin pelimoottorin ulkopuolelta (Sweetser & Wiles, 2002). Skriptaus kieliä voi olla joitain yleisiä, kuten Unityn käyttämä C# -kieli (Unity, 2020), tai ne voivat olla pelimoottorien omia ohjelmointikieliä, kuten Quakesissa QuakeC tai Unrealissa UnrealScript. Skriptauksen etuna on se, että sitä voi aika pienellä kynnyksellä käyttää myös sellaiset henkilöt, joilla ei ole paljoa ohjelmointitaitoja. Esimerkiksi suunnittelijat voivat skriptauksen avulla implementoida peliin tarinan. Lisäksi pelin julkaisun jälkeen myös pelaajat tai harrastelijat voivat itse luoda peliin erilaisia modeja, jos skriptaukset ovat julkisia. Huonona puolena skriptauksessa on se, että se on deterministinen ja vaatii peliohjelmoijan niihin asioihin, jotka pitää kovakoodata. Tällaisia asioita ovat esimerkiksi hahmon liikkuminen ja erilaiset skenaariot, joita pelissä voi tulla vastaan sekä ne tilanteet, joihin pelaaja voi joutua pelissä (Sweetser & Wiles, 2002).

Skriptausta voidaan käyttää monenlaisissa peleissä ja se voi olla todella hyödyllinen työkalu eri genreissä. Yleisin käyttökohde on erilaisten tapahtumien, vastustajan tekoälyn luominen tai tarinankerronta yksinpelattaviin peleihin. First-person-shooting peleissä voidaan niin ikään skriptauksen avulla luoda vastustajille tekoälyä. Näiden lisäksi skriptausta käytetään myös reaaliaikaisissa strategiapeleissä määrittelemään loitsujen toimimisen, tehtäviä tai tarinaa. Skriptaus on myös todettu toimivaksi massiivisissa online moninpeleissä. Skriptausta yleisestikin käytetään paljon, jos ei koko tekoälyn luomiseen niin ainakin jonkin osan tuottamiseen. Esimerkiksi BioWare on käyttänyt skriptausta apunaan monissa peleissään onnistuneesti (Sweetser & Wiles, 2002). BioWaren pelejä ovat esimerkiksi Star Wars, Dragon Age ja Mass Effect -pelit. Aikoinaan monissa taistelupeleissä tekoäly on ennalta skriptattu, eli samankaltaisissa taistelutilanteissa tekoäly toimii aina samalla tavalla. (Yamamoto & all, 2013).

2.2.7 Muut teknologiat

Näiden tekniikoiden lisäksi on olemassa myös monia muita teknologioita, joita itse pelaajat tai tekoälyä tutkivat henkilöt ovat kehittäneet. Esimerkiksi suosittu StarCraft pelin luoja järjestää kilpailun, jossa pelin pelaajat voivat itse lähteä kehittämään voittavaa tekoälyä (Robertson & Watson, 2014). Akateemisissa tutkimuksissa käytetään yleensä uusia ja monimutkaisempia tekoälyjä, joista voisi olla hyötyä myös pelipuolella niin, että saadaan peleihin älykkäämpiä tekoälyjä. Kuitenkin useasti pelipuolella käytetään näitä vanhempia ja yksinkertaisempia teknologioita, joissa määritellään ja manuaalisesti koodataan tekoälyn käyttäytyminen, kuin että käytettäisiin tekoälyssä oppimis- tai päättelytekniikoita. Tähän on useita syitä, mutta yksi tärkeimmistä on se, että peleissä halutaan tekoälyn olevan sellainen, että se on tarpeeksi haastava pelaajalle, mutta kuitenkin voitettavissa, kun taas akateemisissa tutkimuksissa halutaan tekoälyn oppivan todella hyväksi ja se voisi yrittää voittaa pelaajan keinolla millä tahansa. Pelialalla ei myöskään ole kysyntää uusille tekniikoille, koska näissä voi olla ongelmana se, ettei niillä osata tehdä sellaista tekoälyä, joka olisi tarpeeksi haastava ja silti hauska vastus pelaajalle (Robertson & Watson, 2014).

Samassa pelissä voidaan käyttää useaa eri teknologiaa luomaan pelin tekoälyä eri tilanteisiin. Esimerkiksi Black & White -pelissä on käytetty skriptausta, decision tree sekä neural networksia ja Half-life -pelissä on käytetty Finite state machinea ja flockingia (Sweetser & Wiles, 2002).

2.3 Tekoälyn animointi

Tekoälyn ei tarvitse olla joka tilanteessa viisas, eikä siihen välttämättä tarvitse lisätä monimutkaisia kognitiivisia malleja, oppimista tai edes geneettistä oppimista vaan sen puutteita voidaan paikata animaatiolla. Oikeaan paikkaan sijoitettu animaatio luo pelaajalle illuusion tekoälyn älykkyydestä. Esimerkiksi eri tunteita voidaan animaatioiden avulla korvata, sen sijaan että niille rakennettaisiin oma iso algoritmi, jonka avulla tekoäly valitsee tunnetilansa. Ja jos tunnetiloja on monta ja niihin vaikuttavia asioita on monta, voi laittaa tekoälyn valitsemaan tunnetilansa sattumanvaraisesti. Tämän ansiosta tekoälyn voi rakentaa niin, että siihen liitetään vain tärkeimmät työkalut, jotta tekoäly toimii halutulla tavalla, eikä vie liikaa muistia (Millington & Fudge, 2009, s. 22–23). Myös esimerkiksi F.E.A.R -pelissä on luotu illuusio siitä, että pelin tekoälyn ohjaamat hahmot olisivat vuorovaikutuksessa toistensa kanssa, vaikka oikeasti ne seuraavat pelin globaalin tekoälyn antamia käskyjä. Tämä illuusion luominen ei kuitenkaan välttämättä sovi kaikille peleille, riippuen niiden teemoista (Agis, Gottifredi & García, 2020).

3. 3D objektit

Tässä kappaleessa käsitellään 3D mallintamisen eri vaiheita ja eri ohjelmia, joilla 3D objekteja voidaan tehdä. Mallintaminen, mappaaminen, teksturointi, muovaaminen, animointi ja renderointi ovat kaikki tärkeässä asemassa, kun halutaan luoda omia pelihahmoja.
Kun luodaan 3D objektia, aluksi hahmo mallinnetaan. Tämän jälkeen tapahtuu objektin mappaus, jotta saadaan laitettua tekstuuri, jonka jälkeen hahmo muovataan ja animoidaan, jotta saadaan luotua erilaisia asentoja eri skenaarioihin tai eri käyttöihin. Viimeisenä, kun objekti on valmis, niin se visualisoidaan maisemaan, joka tapahtuu renderoinnilla (Gonzalez-Garcia, 2015).

3.1 3D Objektin luominen

3D objekteja voidaan luoda niihin soveltuvilla ohjelmilla. Tällaisia ohjelmia ovat muun muassa Maya, Mudbox, 3Ds Max, Blender, Zbrush. Jokaisella ohjelmalla on omat vahvuutensa ja heikkoutensa tai ovat erikoistuneet tiettyyn asiaan 3D objektien luomisessa (Wallin, 2015, s. 4-5). Kuvan veistämisen ja mallintamisen avulla voidaan luoda 3D objekteja. Tavat ovat hyvin erilaisia ja sopivat eri tavalla eri asioiden luomiseen. (Taylor, 2016). Mallintamista voidaan kutsua myös termillä polygoninen mallintaminen (Wallin, 2015, s.6).

Mallintamisessa työskennellään erilaisten polygonien kanssa, joita lisätään erilaisia verteksejä edgejä ja faceja, joita myös liikutellaan niin että objektista tulee entistä tarkempi ja halutun näköinen. Mallintaminen sopii hyvin sellaisiin 3D objekteihin, joissa on kovia muotoja; ihmisten keksimiin asioihin, kuten erilaisiin koneisiin, aseisiin, huonekaluihin (Taylor, 2016). Tietokoneiden on nopeampi käsitellä mallintamisella tehtyjä 3D objekteja. Objektit voidaan jakaa joko high polygoneihin tai low polygoneihin, riippuen siitä kuinka tiheästi polygoneita on objektissa. Videopeleissä yleensä pyritään low poly objekteihin, jotta saadaan säästettyä tietokoneen tehoja (Wallin, 2015, s. 6).

Low poly mallintamista suositaan videopelien tekemisessä. Termistä tulee mieleen, että objektissa tulisi olla mahdollisimman vähän faceja, mutta todellisuudessa facejen määrä riippuu siitä, mille platformille peliä tehdään. Tietokoneille suunnatuissa peleissä ideaalimäärä vaihtelee 1500 ja 4000 polygonin välillä. Jotta objekti olisi mahdollisimman helppo saada “luihinsa” ja helppo animoida, se tulisi mallintaa mahdollisimman pitkään nelisivuisten polygonien avulla. Mallintamisessa ei tule objektiin lisätä yksityiskohtia, sillä ne voidaan lisätä teksturoinnin yhteydessä ja ne vain vaikeuttaisivat animointia (Wallin, 2015, s. 18).

Kuvan veistämisessä käytetään erilaisia siveltimiä, joiden avulla 3D objekti luodaan. Tällä tavalla saadaan pehmeitä muotoja, jonka takia kuvan veistäminen on hyvä muoto orgaanisille muodoille, kuten hahmoille ja niiden vaatetuksille, eläimille ja hirviöille. Tämä tapa on todella nopea ja intuitiivinen tapa tehdä objekteja sekä luoda näille yksityiskohtia. Haasteina kuitenkin on, että yksittäisten komponenttien muuttaminen ja veistämisellä on vaikea tehdä teräviä reunoja (Taylor, 2016).

Oli tekniikka kumpi tahansa, 3D mallista saa todella yksityiskohtaisen halutessaan. Vaikka nykyään tietokoneet jaksavat pyörittää yli tuhannesta polygonista koostuvaa 3D objektia, niin on hyvä muistaa, että vähempi on parempi. Monet mallintajat joutuvatkin usein tasapainottelemaan objektin yksityiskohtien ja tekstuurin resoluution kanssa, jotta

objekti ei olisi liian raskas pyöritettäväksi reaaliajassa. Usein artistit tekevät objektiin aluksi perusrakenteen, josta lähtevät veistämään objektista todella yksityiskohtaista työtä, jopa niinkin yksityiskohtaista, että objekti on high poly tyyppinen. Tämän jälkeen tapahtuu hard surface mallintaminen, jossa esimerkiksi hahmolle mallinnetaan vaatteet ja varusteet valmiiksi. Kun objekti on valmis, siihen käytetään vielä retopologize nimistä työkalua, jonka avulla high poly –mallista saadaan tehtyä low poly –versio (Wallin, 2015, s. 10).

3.2 Mappaus ja teksturointi

Teksturoinnin avulla 3D objektiin saadaan lisää elävyyttä ja yksityiskohtia. Teksturointi on yksi tärkeä osa, kun peliin luodaan 3D objekteja. Huonosti toteutettu teksturointi tekee objektista raskaan (Wallin, 2015, s. 10). Ennen teksturointia 3D objektista luodaan mappauksen avulla 2D kuva eli bittikartta, koska teksturointi tapahtuu aina 2D tilassa. Mappaukseen on olemassa useita eri tekniikoita, mutta yksi suosituimmista on UV mappaus, jota käytetään esimerkiksi Maya ja Blender -ohjelmissa (Bhagyashree Rout, 2019). Vaikka UV mappaus on nopea ja tehokas mappaus tapa, monet artistit saattavat myös itse mapata objektin. Tämä siksi, koska UV mappaus ei välttämättä aina vastaa sellaista mappausta, jota tarvittaisiin.

Kun 3D objektista on tehty bittikartta, voidaan alkaa luomaan itse tekstuuria. Teksturoinnin tulee olla joskus todella yksityiskohtaista, sillä sen avulla voidaan piilottaa geometrian alhainen resoluutio (Wallin, 2015, s. 10). Teksturointi osuudessa 3D objekti tulee eläväisemmäksi, koska tässä vaiheessa määritellään käytettävän materiaalin sileys, kovuus, kiilto ja kuviot. Teksturoinnissa tulee ottaa huomioon materiaali, joka tulee objektiin, sillä esimerkiksi materiaalin kovuus tai pehmeys tulee vaikuttamaan siihen, kuinka valo heijastuu objektista. Teksturoinnissa voi myös muovata sitä, kuinka valoefektit näkyvät objektissa. Vasta viimeisenä, kun on teksturointi ja valoefektit saatu tehtyä, kannattaa 3D objektille alkaa luomaan yksityiskohtia (Bhagyashree Rout, 2019). 3D objekteissa voi olla useita teksturointi mappeja, joiden avulla saadaan objektiin yksityiskohtia, ettei tarvitsisi tehdä todella yksityiskohtaista objektia (Wallin, 2015, s. 10). Tällaisia ovat esimerkiksi hahmon arvet Teksturointiin on myös olemassa omia ohjelmia, jotka käyttävät PBR systeemiä (Physically based rendering). Jotkin teksturointiohjelmat keskittyvät käyttämään retopology työkalua, jonka avulla voidaan vähentää 3D mallista polygonien määrää ja lisätä edgeflowia. Yleisempiä teksturointiojelmia ovat esimerkiksi Substance Painter, Mari ja BodyPaint 3D (Wallin, 2015, s. 5).

Teksturoinnin jälkeen 3D objektiin lisätään shaderi, jonka avulla yleensä nähdään se, kuinka tekstuuri toimii objektissa. Jokaisella tekstuurilla on omanlaiset shaderit omine arvoineen ja niitä käytetäänkin sen takia, koska niiden avulla on helppoa kontrolloida erilaisia parametreja, kuten esimerkiksi valoisuutta. Shadereita käytetään paljon myös pelimoottoreissa. Shadereissa on useita eri kanavia ja erilaisia mappeja. Näistä tärkeässä asemassa ovat “diffuse map”, “normal map” ja “specular map”. Diffuse mapin avulla määritellään objektin väri. Normal mapin avulla voidaan luoda yksityiskohtia objektiin ja specular mapin avulla hahmotellaan objektin kiiltoa ja korostuksia (Wallin, 2015, s. 13-15).

3.3 Muovaaminen

3D objektin muovaamisella tarkoitetaan esimerkiksi hahmon asennon vaihtamista toiseen. Myös hahmon animointi voidaan toteuttaa tässä työvaiheessa.

Muovaustavoista häkkipohjainen (cage-based) metodi on ollut suosittu tapa, koska se on yksinkertainen ja nopea. Nämä häkkipohjaiset metodit käyttävät yhtä häkkiä, joka ympäröi mallin, jotta sitä voi muotoilla. Häkillä tulee olla samanlainen muoto kuin syöttöverkolla, mutta se on kuitenkin paljon yksinkertaisempi. Tämä häkki ohjaa lopullista muodonmuutosta.

Häkkiverkossakin on olemassa useita eri tapoja. Yksilöhäkissä (single cage) käytetään vain yhden tyyppistä koordinaatiota. Tämä tekee muovauksista globaaleja. Haittana tässä tavassa on, ettei silloin voi yhdistää toisten koordinaattien vahvuuksia ja joustavuutta. Tässä tavassa ei myöskään voi tehdä hahmoon eritasoisia yksityiskohtia. Toisaalta taasen usean häkin käytössä tulee häkkien rajoille epäjatkuvuutta ja tapa eliminoi syöttöverkon sileyden ja saattaa aiheuttaa jopa halkeamia malliin (Gonzalez-Garcia, 2015).

Voidaan myös käyttää luurankoa, jonka avulla taivutellaan ja käännellään hahmo haluttuun asentoon. Luurangon käyttö voi olla yksinkertaista tai sen voi tehdä myös monimutkaiseksi tarvittaessa. Luuranko on helppo asetella hahmoon, mutta vaikeaa on liikkeiden suunnittelu, minkä takia onkin tärkeää, että luuranko on mahdollisimman joustava (Wallin, 2015, s. 18).

3.4 Animointi

Kun tehdään esimerkiksi hahmon animaatiota 3D peliin, tulee ottaa huomioon se, että kyseistä animaatiota voidaan katsoa monesta eri kulmasta. Toisin sanoen, vaikka animaatio näyttäisi hyvältä edestäpäin, se voi näyttää sivulta ja takaa huonolta, jos animointi hetkellä ei ole otettu huomioon kaikkia mahdollisia kuvakulmia (Wallin, 2015, s. 19).

3.5 Renderointi

On olemassa monia erilaisia 3D näkymien renderointi tapoja, mutta suosituimmat tavat ovat Säteenseuranta, joka on kuvapohjainen algoritmi sekä rasterointi, joka on oliopohjainen tekniikka (Xing, 2020). Säteenseurannan etuna on se, että sen avulla voidaan renderoida melko tarkasti realistisia kuvia, mutta reaaliaikainen fotorealistinen grafiikka ei ole sillä vielä mahdollista (Wikipedia, 2019) Toinen etu on se että Säteenseuranta voidaan rinnakkaistaa muuhun tietokoneen toimintaan helposti, koska sen algoritmi skaalautuu lineaarisesti käytettävissä olevien prosessorien määrään (Nery, Nedjah & França, 2016) Säteenseuranta -algoritmi jäljittää polun kullekin valonlähteestä lähtevästä valonsäteestä sinne asti, kunnes säde osuu malliin ja siitä kameran linssiin (Nery & all., 2016) Säteenseuranta tapa on hidasta, koska valonsäteitä ja heijastuksia voi olla paljon, jolloin jokainen pikseli joudutaan tarkastelemaan vuorotellen ja jokaiselle pikselille katsotaan ne oliot, jotka vaikuttavat pikseliin. Tämän jälkeen pystytään laskemaan pikselin arvo (Nery & all., 2016. Xing, 2020). Tavan hitaudesta huolimatta Nvidia on kehittänyt oman säteenseuranta -algoritmin, jota on käytetty muun muassa Battlefield V (heijastukset), Metro Exodus (Globaali illuminaatio), Shadow of the Tomb Raider (varjot), Stay in the Light (Globaali illuminaatio ja heijastukset) sekä Quake II

RTX (Globaali illuminaatio) (Thomas & Hayward, 2019). Säteenseuranta tapaa ovat käyttäneet esimerkiksi ammattikäyttöön tarkoitettu 3D mallinnusohjelmat Maya ja Autodesk 3da Max sekä suomalaisten kehittämä Realsoft 3D (Wikipedia, 2019)

Rasterointi -tekniikka on oliopohjainen (object-order) renderointi tekniikka. Siinä käydään vuorollaan jokainen objekti läpi. Myös objektin kaikki pikselit käydään lävitse niin, että niiden vaikutukset muihin pikseleihin löydetään ja läpikäytävän pikselin arvoa päivitetään. Rasterointi -tekniikan etuna on sen renderointi tehokkuus (Xing, 2020).

4. Pelimoottorit

Tässä kappaleessa käsitellään pelimoottoreita. Pelimoottorit ovat huomattavasti helpottaneet pelien luomista ja niiden avulla voidaan luoda graafisesti hienoja pelejä, joiden FPS on kuitenkin matalalla. Pelimoottoreita on tarjolla erilaisia, joista osa on maksullisia ja osa ilmaisia tietyin lisenssi ehdoin.

4.1 Yleisesti pelimoottoreista

Pelimoottorit ovat mahdollistaneet sen, että videopeliala on kasvanut huomattavasti uusien videopelejä tekevien yritysten myötä (Torres-Ferreyros, Festini-Wendorff & Shiguihara-Juarez, 2017). Nykyään onkin tavallista, että jokaisen pelin pohjalla toimii jonkinlainen pelimoottori, joka käsittelee pelin logiikkaa ja etenkin dataa, jota pelissä käytetään. Tämä data voi olla esimerkiksi pelilogiikka, pelin grafiikat ja musiikki. (Rojas, 2013). Pelimoottoreiden edut ovat niiden uudelleenkäytettävissä koodeissa ja kirjastoissa, jotka on luotu juuri pelien tekemistä ajatellen niin ettei ohjelmoijan tarvitse keskittyä teknisiin yksityiskohtiin peliä luodessa. Pelimoottoreissa on oliopohjainen ohjelmointi tapa, jonka avulla voidaan luoda luokkia perinteiseen tapaa, mutta se on intuitiivisempi. Lisäksi pelimoottoreihin on ohjelmoitu prosessi, jossa monimutkainen algoritmi laskee valmiiksi valot ja varjot, joita peli tarvitsee (Torres-Ferreyros, Festini-Wendorff & Shiguihara-Juarez, 2017). Jotkin pelimoottorit on voitu suunnitella vain tiettyä peligenreä ajatellen, mutta yleisesti ottaen on olemassa paljon yleispelimoottoreita, joilla voi tehdä kaikenlaisia pelejä (Rojas, 2013). Nykyaikaisia pelimoottoreita ovat Game Maker, JMonkey, Marmalade, Ogre3D, Shive, Sio2, Turbulenz, Unreal Engine 4 ja Uniy3D, joista kaksi viimeisintä ovat suosituimpia pelimoottoreita (Chursin & Semenov, 2019). Näissä pelimoottoreissa on alhainen liittymämaksu tai ne voivat olla täysin ilmaisia. Esimerkiksi Unreal Engine veloittaa 5 % pelin myynnistä markkinoilla (Torres-Ferreyros, Festini-Wendorff & Shiguihara-Juarez, 2017). Seuraavissa kappaleissa käydään lävitse pelimoottoreiden yleisimpiä ominaisuuksia ja tarkastellaan millaisia ne ovat Unity3D ja Unreal Engine -pelimoottoreissa. Hyvässä pelimoottorissa olisi hyvä olla renderointi, törmäyksen havaitseminen, fyysisyys, tekoäly, skriptaus ja editori, jolla pelinkehittäjä voi muokata peliä. Editoriin kuuluu erilaiset shaderit, valoisuusasetukset, tekstuurit ja materiaalit, joiden avulla voidaan luoda pelimaailmoja.

4.2 Renderointi videopelissä

Renderointi pelissä on yleensä reaaliaikaista ja monissa pelimoottoreissa onkin panostettu tähän algoritmiin niin, että peliin saadaan elokuvatason grafiikkaa reaaliajassa (Rojas, 2013). Esimerkiksi Unity3D pelimoottoreissa on erilaisia renderointi pipelineja, kuten Built-in Render Pipeline, Universal Render Pipeline (URP), The High Definition Render Pipeline (HDRP) ja lisäksi Unity tarjoaa pelintekijälle mahdollisuuden luoda oman renderointi pipelinen (Scriptable Render Pipeline eli SRP) Kaikki ovat Unityn tekemiä. Näistä ensimmäinen on Unityn perusrenderointi pipeline, joka on suunniteltu yleiskäyttöön ja sitä voi muokata vain vähän haluamakseen. URP ja HDRP ovat molemmat esikäännetty SRPsta. URP on suunniteltu artistiystävälliseksi niin, että sillä voi optimoida grafiikat eri laitteille helposti ja nopeasti. HDRP on suunniteltu niin. että sillä saa aikaan todella korkealaatuista grafiikkaa. HDRP käyttää fyysiseen perustuvaa valaistusta ja materiaalia, sekä tukee eteenpäin ja lykättyä renderointia. SRP sallii

käyttäjän kontrolloivan renderointia C# skriptien avulla (Unity, 2020). Unreal Engine 4 käyttää renderoinnissa DirectX 11 pipelineä, jonka pääominaisuudet ovat lykätty varjostus, globaali illuminaatio, valaistuksen läpinäkyvyys ja efektien jälkikäsittely (Epic Games, 2020).

4.3 Törmäyksen havaitseminen

Törmäyksen havaitsemista pidetään lähes yhtä perusmekanismina kuin renderointia ja se yleensä onkin osa fyysistä puolta. Törmäyksen havaitseminen peleissä vaatii reaaliaikaisen havaitsemissysteemin, joita onkin kehitelty muutamia erilaisia, kuten diskreetti ja jatkuva törmäyksen havaitseminen, rajaava volyymi ja alueellinen osiointi. Diskreetti ja jatkuva törmäyksen havaitseminen on hyvin tarkka mutta se on myös laskennallisesti kallis. Tämä johtuu siitä, että algoritmin ottaa huomioon osallistuvien elinten liikkeet, jotta se löytäisi mahdollisimman tarkan vaikutusajan näiden elinten välille. Rajaava volyymi -tapa on yleinen peleissä ja sitä käytetään törmäyksen havaitsemisessa jo ennen kuin testataan objektin omaa muotoa törmäyksen havaitsemiseen. Rajaavassa volyymissa on kaksi erilaista tapaa, joista suositumpi on axis-aligned bounding boxes (AABB) ja sitten on oriented bounding boxes (OBB). Alueellisessa osioinnissa erityisiä tietorakenteita, joihin tallennetaan tietoa alueesta, jossa kohtauskohteet ovat sillä hetkellä ja jätetään huomiotta ne objektit, jotka ovat liian kaukana. Tätä tapaa on käytetty myös renderointimoottoreissa niin etteivät kyseiset moottorit prosessoi niitä asioita, jotka eivät näy kamerassa (Roja, 2013).

Unity käyttää omassa ohjelmassaan jatkuvaa törmäyksen havaitsemista, josta on olemassa variaatiot pyyhkäisypohjainen (sweep-based) ja spekulatiivinen. Pyyhkäisypohjaisessa tavassa käytetään Time Of Impact -algoritmia laskemaan mahdolliset törmäykset objektille niin että algoritmi pyyhkäisee objektin eteenpäin sen nykyisellä nopeudella ja jos edessä on jotain, mihin törmätä, algoritmi laskee, milloin isku tapahtuu ja siirtää objektia siihen saakka, kunnes kyseinen ajankohta käy toteen. Ongelmana tässä tavassa on se, että lineaarinen pyyhkäisy jättää huomiotta objektin kulmaliikkeet, mikä voi aiheuttaa sen, että objekti menee törmättävästä kohteesta läpi, jos objekti pyörii samalla kun se liikkuu. Toinen ongelma tässä tavassa on se, että jos pelissä on iso määrä nopeasti liikkuvia objekteja lähekkäin, niin törmäyksen havaitsemisohjelma ylikuormittuu nopeasti ylimääräisten pyyhkäisyjen takia. Spekulatiivisessa tavassa käytetään AABB-pohjaista tapaa törmäyksen havaitsemiseen. AABB perustuu objektin lineaariseen liikkeeseen ja kulmaliikkeeseen ja kyseisessä tavassa algoritmi tarkistaa kaikki sellaiset kontaktit, jotka voivat olla mahdollisia seuraavassa fyysisessä askeleessa. Tämän takia objekti ei tunkeudu toisesta objektista läpi törmäyksen aikana. Spekulatiivinen tapa on parempi ja halvempi kuin pyyhkäisy, mutta ei siltikään täysin ongelmaton. Tässä tavassa voi tapahtua haamutörmäys, missä liikkuvan objektin liikkumissuunta voi yllättäen muuttua (Unity, 2018).

Unreal Enginessä ei tarkalleen kerrota, minkälaista törmäyksen hallinnan algoritmia he käyttävät. Tässä pelimoottorissa käyttäjä voi itse määritellä, kuinka törmäyksen hallinta toimii missäkin objektissa. Vaihtoehtoina ovat huomiotta jättäminen, päällekkäisyys ja blokkaaminen (Epic Games, 2020).

4.4 Fyysisyys

Fysiikalla tarkoitetaan yleensä klassista mekaniikkaa pelimoottoreissa. Moni pelifysiikkamoottori sisältää jäykän kappaleen (rigid body), liitokset (joints) ja räsynuket (ragdolls). Vaikka esimerkiksi jäykille objekteille ja pehmeille objekteille koitetaan luoda niille ominaista normaalia käyttäytymistä, niille on olemassa omia spesiaaleja komponentteja, joiden avulla voidaan rikkoa joitakin fysiikan lakeja tietyissä tilanteissa. Saatavilla olevia pelifysiikanmoottoreita ovat Box2D ja Nape 2D peleille sekä ODE, Havok, PhyX ja Bullet moottoreita käytetään 3D peleissä (Roja, 2013).

Unityssa fysiikan toteuttamiseen on erilaisia tapoja, riippuen millaiseen projektiin käyttäjä tarvitsee fysiikkaa. Oliopohjaisissa projekteissa Unitylla on sisäänrakennettu fysiikkamoottorit PhysX 3D peleille ja Box2D 2D peleille. Datapohjaisiin projekteihin käyttäjän tulee asentaa erikseen Unity Physics tai Havok Physics for Unity -paketit (Unity, 2020). Unreal Engine 4 käyttää myös PhysX fysiikkamoottoria (Epic Games, 2020).

4.5 Tekoäly

Tätä aihetta on käsitelty jo luvussa 3 aika tarkasti. Pelimoottoreissa voi olla valmiiksi implementoituna joitain tärkeitä funktioita, joiden avulla voidaan luoda peliin tekoälyä. Esimerkiksi Unreal Enginessä ja Unityn Pro versiossa on sisäänrakennetut polunetsintä funktiot (Roja, 2013). Tämä navigointi tapa antaa ei-pelattavalle hahmolle mahdollisuuden liikkua itsekseen pelimaailmassa. Unitylla on kehitteillä oma Machine Learning agentti, mutta se on vielä beta -vaiheessa (Unity, 2020). Unreal Enginessä voi myös luoda omaa tekoälyä, joka pohjautuu Behavior Trees algoritmiin. Se saa informaatiota ympäristöstä Environment Query Systemsin avulla. AI Perception järjestelmää käyttäessä, tekoäly saa tietoa esimerkiksi äänistä, näkyvyydestä ja vahingoittumisesta. Unreal Enginessä, kun käyttää kaikkia 3 järjestelmää tekoälyä luodessa, niin voi saada aikaan todella hyvän tekoälyn (Epic Games, 2020).

4.6 Skriptaus

Skriptaus auttaa erottelemaan pelin omat säännöt pelimoottorista. Pelimoottori lataa skriptatut säädökset ja pyörittää niitä, kun peliä ajetaan (Roja, 2013). Monet pelimoottorin applikaatiot tarvitsevat skriptejä, jotta voivat vastata pelaajan antamaan syötteeseen. Skriptejä voi käyttää luomaan graafisia efektejä, kontrolloimaan fyysistä käyttäytymistä, kuten liikkumista, ja skriptien avulla voi muokata tekoälyä haluamakseen (Unity, 2020). Joissakin pelimoottoreissa voi olla omia skriptaus kieliä tai ne voivat käyttää standardeja skriptaus kieliä. Unity käyttää esimerkiksi Monoa ja sallii skriptauksen C# kielellä, UnityScriptillä ja Boolla (Roja, 2013). Unitylla on erilaisia työkaluja, jotka auttavat skriptien luomisessa. Näitä ovat konsoli. log -tiedostot, Unity Test Framework, jolla voi testata omaa koodia sekä muokkaus moodissa että pelimoodissa, C# kääntäjä IL2CPP, joka on Unityn kehittämä skriptauksen backend, jota voi käyttää Mono kielen sijaan, jos kehittää projektia toisella alustalla. Lisäksi Unity asentaa Visual Studiosin, kun käyttäjä asentaa pelimoottorin koneellensa (Unity, 2020). Unreal Engine 4 tarjoaa kaksi työkalusarjaa, joita voi käyttää myös yhdessä. Unreal Engine 4 koodia voi kirjoittaa C++ -kielellä tai käyttää visuaalista skriptausta eli blueprinttejä. C++ luokkia voi käyttää pohjana blueprinttien luokille. Koodia voi kirjoittaa joko Visual Studiolla tai XCodella, jotka Unreal Editor tarjoaa (Epic Game, 2020).

4.7 Editori

Unity3D pelimoottori tukee C# ohjelmointikieltä, sillä voi tehdä sekä 2D että 3D -pelejä. Lisäksi Unity3Dllä on joustava lisenssikäytäntö (Chursin & Semenov, 2019). Esimerkiksi yksittäisille ihmisille on olemassa ilmaiset versiot erikseen opiskelijoille ja muille. Opiskelijoille Unity3D ohjelmassa on hieman enemmän ominaisuuksia kuin henkilökohtaisessa lisenssissä. Henkilökohtaisella lisenssillä saa pelillä tienata alle 100 000 dollaria viimeisten 12 kuukauden aikana ilmaiseksi. Liiketoimintaa harjoittaville Unity tarjoaa 3 erilaista sopimusta: plus, pro ja yritys (Unity, 2020). Editorin avulla voidaan visuaalisesti luoda pelimaailmaa, hallita kaikkea peliin tulevia asioita. Esimerkiksi Unityn editorissa voidaan luoda erilaisia scenejä ja hallita malleja, tektuureita, animaatiota, shadereita, skriptejä ja ääni- sekä musiikkitiedostoja. Editoreiden avulla pelien tuottamisesta on tullut helpompaa (Rojas, 2013).

5. Keskustelu

Tämän kirjallisuuskatsauksen tulokset antavat yleiskäsitystä siitä, mitä erilaisia tekoälyjä peleille on olemassa, 3D mallinnuksen perusteet yleisesti ja mallintamiseen käytettyjä yleisiä ohjelmia sekä pelimoottoreiden rakenteen ja miten ne ovat helpottaneet pelien luontia. Tässä kappaleessa käymme tutkimuskysymykset ja niiden vastaukset lävitse

Mitä eri tekoälyjä on peleille olemassa?
Pelien tekoälyjä on monia erilaisia ja osa niistä ovat vanhoja, mutta todella toimivia. Finite State Machine on yksi suosituimmista teknologioista, koska se on helppo ohjelmoida ja sitä on helppo ymmärtää. Skriptaus on myös suosittu tapa ja se sopii eri peligenreihin ja sen avulla voidaan tehdä monimutkaiset asiat yksinkertaisemmin. Myös Neural Networkkia ja sen muunnoksia sekä Fuzzy Logicia käytetään pelien tekoälyn luomiseen. Vanhojen hyvien tekniikkojen lisäksi myös joitain uusia tekniikoita löytyy, Näistä ylivoimaisesti suosituin on Behavior Trees, joka on helppo suunnitella, debugata ja testata. Lisäksi kyseisessä teknologiassa voidaan ensin ohjelmoida tekoälylle toimintoja ja vasta sen jälkeen suunnitella puun rakenne. Jokaisella teknologialla on omat vahvuutensa ja heikkoutensa, ja ne voivat sopia yhteen tai useampaan eri peligenreen. Vaikka vanhat teknologiat ovat luotettavia, edelleen etsitään uusia ja parempia tapoja pelien tekoälyn kehittämiseen. Tekoälyn kehittämisestä järjestetään myös kilpailuja, joissa osallistujat koittavat kehittää sellaisen tekoälyn, joka päihittää pelin tekoälyn. Nämä uudet tekoälyt voivat olla erilaisia hybridejä tai kehittyneempiä tekoälyjä. Myös akateemikot kehittävät peleihin tekoälyjä, mutta yleensä pelifirmat käyttävät tuttuja tekoälyjä, koska niiden toimivuus tiedetään ja tekoälystä saa tarpeeksi haasteellisen pelaajille ja kysyntää uusille tekniikoille ei ole paljoa.

Mitä eri työkaluja on olemassa 3D objektien luomiseen?
3D objektien luomiseen on olemassa useita erilaisia ohjelmia ja ne painottuvat eri asioihin. Tunnetuimpia ohjelmia ovat Maya, Zbrush ja Blender. Lisäksi löytyy Max ja Mudbox Se mikä ohjelma sopii itselleen, riippuu siitä, mitä vaatimuksia on ohjelmalle. Esimerkiksi Zbrush ja Mudboxovat painottuneet objektien muovaamiseen, kun taas Blender on hyvä yleisohjelma mallintamiselle ja siitä löytyy perusasiat koko 3D mallintamiselle, mutta on painottunut renderointiin silti eniten. Myös teksturointiin on olemassa omia ohjelmia, kuten Substance Painter, Mari ja BodyPaint 3D sekä monia muita ohjelmia

Kuinka luodaan 3D objekteja?
3D objektien luomisessa on useita vaiheita, jos halutaan luoda toimiva ja animoitu objekti. Kaikki lähtee mallintamisesta ja veistämisestä. Videopeleissä suositaan lowpoly hahmoja, jotka luodaan tekemällä aluksi highpoly objekti, joka mapataan, teksturoidaan, laitetaan eri asentoihin ja animoidaan, jonka jälkeen retopologize -työkalua. Tämä työkalu high poly -malleista saadaan low poly -versioita. 3D hahmoille voidaan ennen tätä mallintaa valmiiksi esim vaatteita ja esineitä.
Mappauksessa suosituin tapa on UV mappaus, mutta jotkut tykkäävät myös itse mapata objektinsa. Teksturoinnin voi tehdä erillisessä ohjelmassa, mutta esimerkiksi Blenderillä voi mallintamisen lisäksi myös teksturoida objektin. Teksturoinnin avulla on tarkoitus luoda 3D objektiin yksityiskohtia ja saada objektille väriä pintaan. Animointiin ja muovaamiseen 3D objekti tarvitsee luurangon tai jonkin häkkiverkon, mutta luuranko on näistä käytetympi tapa nykyisin. Videopeleissä on enemmän myös alettu keskittymään hyvään animaatioon, jopa niinkin tarkasti, että hahmon naaman liikkeitäkin on alettu animoimaan peleihin.

Millaisia pelimoottorit ovat ja miten ne auttavat pelien luomisessa?
Pelimoottorit ovat työkaluja, joiden avulla voidaan luoda videopelejä aiempaa helpommin. Niissä on useita hyödyllisiä ominaisuuksia, joiden avulla voi visuaalisesti muokata pelimaailmaa. Mahdolliset 3D objektit niihin pitää kuitenkin luoda itse tai voi ostaa pelimoottorien tekijöiden kaupasta, jos sellaista on. Pelimoottoreissa on monia hyviä ominaisuuksia; renderointi, törmäyksen havaitseminen, fyysisyys, skriptaus mahdollisuus, mahdollinen tekoäly ja editori, jonka avulla hallitaan koko pakettia. Tunnetuimmat pelimoottorit ovat Unity3D ja Unreal Engine, mutta näiden lisäksi on olemassa tai tekeillä open source software pelimoottoreita. Yksi suuri etu pelimoottoreissa on niiden uudelleenkäytettävyys ja helppous.

Renderointi on peleissä reaaliaikaista ja monissa pelimoottoreissa tähän on panostettu niin paljon, että pelimoottori voi pyörittää elokuvatason grafiikkaa. Fyysisyyden ja törmäyksen havaitsemisen avulla peliin saadaan mukaan realistisuutta niin ettei pelaaja pysty esimerkiksi juoksemaan ison kiven tai kallion läpi, jos molempiin on asennettu rigidbody. Pelimoottoreihin on myös valmiiksi implementoitu sellaisia funktioita, joiden avulla saadaan luotua tekoälyä peliin. Skriptauksen avulla voidaan luoda omia sääntöjä peliin, kuten esimerkiksi kuinka pelaaja liikkuu pelissä. Editorin avulla pidetään koko pakettia kasassa ja se onkin yksi sellainen osa pelimoottorissa, joka on helpottanut pelien tekemistä.

Se mikä ohjelma sopii 3D mallintamiseen, pelien tekemiseen tai mikä tekoäly sopii juuri siihen peliin, riippuu paljon siitä, mihin käyttäjä on tottunut, onko projektissa mukana paljon vai vähän ihmisiä. Esimerkiksi Unity3D sopii hyvin pienille porukoille ja Unreal Enginea voi hallita paremmin isompikin porukka. 3D ohjelmissa taasen riippuu siitä, mihin haluaa panostaa mallintamisessa.

Jatkotutkimuskohteita voivat olla erilaiset vertailut tekoälyjen, 3D ohjelmien tai pelimoottoreiden välillä. Voisi myös tehdä tutkimusta siitä minkälainen tekoäly voisi sopia mihinkin peligenreen. 3D mallintamisessa voisi UV mappauksen algoritmia kehittää tai syvemmin paneutua retopologize työkalun toimintaan. Pelimoottoreissa voi kehittää renderointia tai muita tärkeitä ominaisuuksia, joita on käyty läpi. Myös 4k tekniikkaa ja renderointia peleissä voisi tutkia.

Lähdeluettelo

Agis, R. A., Gottifredi, S., & García, A. J. (2020). An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, 155 doi:10.1016/j.eswa.2020.113457

Alan, T. (2011) *UDK Game Development.* Boston, Mass: Course Technology PTR

Bhagyshree Rout (19.8.2019) *The Tecnique of Texturing* [Blogikirjoitus]. Haettu osoitteesta http://www.videogyan.com/blog/stories/the-technique-of-texturing/

Chursin, G., & Semenov, M. (2020). Learning game development with Unity3D engine and arduino microcontroller. Paper presented at the *Journal of Physics: Conference Series, , 1488*(1) doi:10.1088/1742-6596/1488/1/012023

Gonzalez-Garcia, F.(2015). *Trends in continuity and interpolation for computer graphics.* IEEE Computer Society 2015, 76-82. doi:10.1109/MCG.2015.129

Taylor, J. (8.8.2016) . *MODELING VS SCULTING: HOW DO YOU KNOW WHICH TO USE?* [Blogikirjoitus]. Haettu osoitteesta https://www.methodj.com/modeling-vs-sculpting/

Lu, F., Yamamoto, K., Nomura, L. H., Mizuno, S., Lee, Y., & Thawonmas, R. (2013). Fighting game artificial intelligence competition platform. Paper presented at the 2013 *IEEE 2nd Global Conference on Consumer Electronics, GCCE 2013*, 320-323. doi:10.1109/GCCE.2013.6664844

Millington, I & Funge, J. (2009) *Artificial intelligence for games*. Elsevier Inc.

Nery, A. S., Nedjah, N., & França, F. M. G. (2016). *A parallel architecture for ray-tracing*. Paper presented at the Proceedings - 2010 1st IEEE Latin American Symposium on Circuits and Systems, LASCAS 2010, 77-80. doi:10.1109/LASCAS.2010.7410224

Newzoo. (2018) Global Games Market Repot. Haettu osoitteesta https://cdn2.hubspot.net/hubfs/700740/Reports/Newzoo_2018_Global_Games_Market_Report_Light.pdf

Robertson, G., & Watson, I. (2014). A review of real-time strategy game AI. *AI Magazine*, 35(4), 75-104.

Rojas, J. (2013) *Getting Started with Videogame Development.* 26th Conference on Graphics, Patterns and Images Tutorials, Arequipa, Peru, 2013, pp. 1-5, doi: 10.1109/SIBGRAPI-T.2013.10.

Sweetser, P. & Wiles, J. (2002) *Current AI in Games: A Review.* Australian Journal of Intelligent Information Processing Systems, 8(1), pp. 24-42.

Thomas, B., Hayward, A., (20.08.2019). *What is ray tracing? The games, the graphics cards and everything else you need to know* [Blogikirjoitus]. Haettu osoitteesta https://www.techradar.com/uk/news/ray-tracing

Torres-Ferreyros, C. M., Festini-Wendorff, M. A., & Shiguihara-Juarez, P. N. (2017). Developing a videogame using unreal engine based on a four stages methodology. Paper presented at the *Proceedings of the 2016 IEEE ANDESCON, ANDESCON 2016,* doi:10.1109/ANDESCON.2016.7836249

Turan, E., & Çetin, G. (2019). Using artificial intelligence for modeling of the realistic animal behaviors in a virtual island. *Computer Standards and Interfaces, 66* doi:10.1016/j.csi.2019.103361

Unity. 2020. Unity User Manual (2019.4 LTS) https://docs.unity3d.com/Manual/index.html

Unreal Engine. 2020. Unreal Engine 4 Documentation. https://docs.unrealengine.com/en-US/index.html

Wallin, K. (2015). *Facial Animation Of Game Characters.* (Opinnäytetyö, Lahden ammattikorkeakoulu)

WePC. 2019 Video Game Industry Statistics, Trends & Data. (2019) https://www.wepc.com/news/video-game-statistics/ .

Wikipedia. (29.04.2019) Säteenseuranta. Haettu osoitteesta https://fi.wikipedia.org/wiki/S%C3%A4teenseuranta

Xing, S., Sang, X., Cao, L., Guan, Y., & Li, Y. (2020). *High-speed integral image object-order rendering method based on reverse perspective technique and optical reconstruction.* Optik, 202 doi:10.1016/j.ijleo.2019.163025