Tampereen ammattikorkeakoulu

Palvelimelta ladattavan tiedon käyttäminen videopelin toiminnan syötteenä

Reino Hämäläinen

OPINNÄYTETYÖ
Elokuu 2019

Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

REINO HÄMÄLÄINEN:
Palvelimelta ladattavan tiedon käyttäminen videopelin toiminnan syötteenä

Opinnäytetyö 25 sivua, joista liitteitä 0 sivua
Heinäkuu 2019

Traplight Games videopeliyhtiö on videopeliprojektissa Battle Legion käyttänyt järjestelmää, jonka avulla pelisimulaation toimintaa voidaan muuttaa ilman että pelisimulaation lähdekoodia käännetään uudestaan. Tämä opinnäytetyö tutkii järjestelmää, kuinka se toimii, mitä hyötyä siitä on ollut pelin kehitystiimille ja mitä ongelmia järjestelmässä on ollut.

Opinnäytetyössä tutkittiin järjestelmän toimintaa. Järjestelmän toiminnan periaatteet on pyritty esittämään selkeästi ja tiiviistetysti. Samoja periaatteita voidaan toivottavasti hyödyntää muissa projekteissa, joissa samanlaiselle järjestelmälle on tarvetta. Järjestelmään perehdyttiin osallistumalla sen rakentamiseen ja ylläpitämiseen.

Opinnäytetyössä todettiin, että järjestelmän toiminnan rakennuspalikat ovat yksinkertaisia. Suuri määrä pelisimulaation toiminnasta määräytyy muokattavan tiedon perusteella, joka on yksi simulaation syötteistä. Tämä järjestelmä vähensi pelijärjestelmän kehittämiseen menevää aikaa, mutta aiheutti myös ongelmia järjestelmän altistamien virheiden takia.

Asiasanat: pelisimulaatio, palvelimet, videopelit

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Production

REINO HÄMÄLÄINEN:
Using Data Provided by a Server as Input in Videogames

Bachelor's thesis 25 pages, appendices 0 pages
July 2019

Traplight Games Ltd. has in their game project Battle Legion used a system, where the functions of the game simulation in the game can be modified without recompiling the simulation's source code. This thesis inspects this system, describes how it works, what utility does it have for the developer team, and what problems have come up during the usage of the system.

This thesis seeks to present an overview of the functionality of the system. This information may be used to help structure a similar system, if such a system is deemed useful for a future project. Information regarding the system is based on the author's experience in building and maintaining the system in question, and on the interviews of the members of the development team.

The thesis concludes that the functionality of the system is based upon quite simple principles. Much of the game simulation's function is determined by data, which is given to the simulation as one of its inputs. The system reduced the time it takes to make meaningful iterations to the simulation in the game, but it also caused problems by introducing errors caused by misuse of the system.

Key words: game simulation, server, video games

SISÄLLYS

1 JOHDANTO 5
2 NYKYISEN JÄRJESTELMÄN TAUSTA 6
2.1 Pelin lyhyt kuvaus 6
2.2 Pelisimulaatio 7
2.3 Projektin tarpeet pelisimulaatiolle 8
3 TOTEUTUKSEN KUVAUS 9
3.1 Yleiskatsaus 9
3.2 Järjestelmän käyttäminen 10
4 JÄRJESTELMÄN TOIMINTA 11
4.1 Simulaatiodatan käyttäminen 11
4.2 Simulaation muuttaminen simulaatiodatalla 12
4.3 Esimerkki järjestelmän toiminnasta 13
4.4 Esimerkki simulaatiodatan muuttamisesta 16
4.5 Järjestelmän rajoitteet 19
5 JÄRJESTELMÄN HYÖDYT 20
5.1 Simulaation muuttamisen helppous 20
5.2 Simulaation toiminnan iteroimisen nopeuttaminen 20
6 KOHDATUT ONGELMAT TOTEUTUKSESSA 21
6.1 Alttius käyttäjien virheisiin 21
6.2 Virheiden huomaamiseen menevä aika 21
6.3 Sisäisen dokumentaation puute 23
6.4 Lisääntynyt koodin haarautuminen 23
7 POHDINTA 24
LÄHTEET 25

1 JOHDANTO

Useissa videopeleissä on järjestelmiä, joiden avulla palvelimelta ladattu tieto muokkaa pelijärjestelmän toimintaa. Traplight Games:in Battle Legion projektissa on käytössä järjestelmä, jonka avulla pelisimulaation toimintaa voidaan hyvin laajamittaisesti muokata muuttamalla taulukkoa, joka ladataan peliä simuloiville tietokoneille. Näin pelin lähdekoodia ei tarvitse kääntää, eli muuttaa tekstimuodosta tietokoneella suoritettaviksi käskyiksi, jotta pelisimulaation toiminnassa voitaisiin nähdä merkittäviä muutoksia.

Tämä opinnäytetyö tutkii kyseistä järjestelmää ja pyrkii selvittämään sen toimintaa. Opinnäytetyössä myös selvitetään järjestelmän luomiseen ajaneita tarpeita. Tämän lisäksi käydään läpi, kuinka järjestelmä on toiminut yhtenä pelin kehitystyökaluista. Järjestelmän hyötyjä, haasteita ja riskejä selvitetään pelin kehittäjien näkökulmasta.

Lopulta tarkoituksena on tutkia, onko järjestelmästä ollut hyötyä. Tämän tiedon oletetaan olevan hyödyllistä kenelle tahansa, joka harkitsee samanlaisen järjestelmän toteuttamista osana uutta peliprojektia.

2 NYKYISEN JÄRJESTELMÄN TAUSTA

2.1 Pelin lyhyt kuvaus

Battle Legion (aikaisemmalta työnimeltään Battle Slides) on Traplight Games videopeliyhtiön peliprojekti. Pelissä pelaajat kokoavat armeijoita asettelemalla yksikköjoukkoja pelikentälle. Nämä armeijat taistelevat toisten pelaajien samalla tavalla kokoamia armeijoita vastaan automaattisesti. Pelaaja on katsojan roolissa itse taisteluiden aikana. Kuva 1 on kuvakaappaus pelissä tapahtuvista taisteluista.

KUVA 1. Kuvakaappaus Battle Legionin taistelusta

Pelikentällä voi olla useita kymmeniä yksittäisiä yksiköitä, jotka etsivät vihollisyksiköitä, aktivoivat animaatiotiloja ja reagoivat pelikentän tapahtumiin. Jokainen yksikkö itsessään on hyvin yksinkertainen kokonaisuus. Pelin yksiköt seuraavat yhtenäisiä sääntöjä. Esimerkiksi kaikki yksiköt pelissä ovat fysiikkasimulaation alaisia ja voivat syttyä palamaan. Eri yksiköiden väliset vuorovaikutukset luovat kuitenkin monimutkaisia ja vaikeasti ennustettavia pelitilanteita.

2.2 Pelisimulaatio

Battle Legion koostuu useasta osasta ja näkymästä. Peli voidaan kuitenkin karkeasti jakaa kahteen osioon.

1. Pelaaja rakentaa oman armeijansa eri pelissä olevista yksiköistä. Kuva 2 esittää näkymän, jossa pelaaja voi rakentaa omaa armeijaansa.
2. Pelaaja taistelee muiden pelaajien rakentamia armeijoita vastaan käyttäen pelaajan rakentamaa armeijaa.

Tässä opinnäytetyössä keskitytään itse taisteluihin, josta käytetään termiä pelisimulaatio. Kaikki muutkin pelin osat ovat osa peliohjelmaa, mutta yleensä viitatessa pelilogiikkaan, viitataan juuri pelisimulaatioon, joka simuloi pelimaailmaa ajassa eteenpäin. Itse simulaation toiminnan selittäminen on tämän opinnäytetyön tavoitteiden ulkopuolella.

KUVA 2. Näkymä, jossa pelaaja rakentaa armeijaansa

2.3 Projektin tarpeet pelisimulaatiolle

Battle Legionin kehityksessä kohdattiin muutamia selkeitä tarpeita, joihin pyrittiin löytämään ratkaisukeino:

1. Pelisimulaation toiminta on oltava helposti muokattavissa pelisuunnittelijan toimesta. Pelin tasapainottaminen vaatii useita iteraatioita, joten kaikki työaika, mikä menee simulaation yksityiskohtien muuttamiseen vaikeuttaa suunnittelutyötä huomattavasti.
2. Peliin pitää pystyä lisäämään yksiköitä, erikoiskykyjä ja muuta lisäsisältöä mahdollisimman helposti. Peliä on tarkoitus kehittää ja ylläpitää pitkän aikaa.

3 TOTEUTUKSEN KUVAUS

3.1 Yleiskatsaus

Jotta Battle Legionin pelisimulaatio voi toimia, tarvitsee se kaksi elementtiä. Ensinnäkin se tarvitsee itse simulaation lähdekoodin. Tämä lähdekoodi on kirjoitettu C#-ohjelmointikielellä, ja se käännetään tietokoneille, jotka suorittavat pelisimulaatiota. Tämä ohjelma sisältää kaiken mitä pelisimulaatio-ohjelman voidaankin olettaa sisältävän. Se määrittelee esimerkiksi fysiikkasimulaation säännöt, funktiot, jotka aktivoivat animaatioita, kuinka vahinko lasketaan yksiköihin ja mitä eri tiloja yksiköillä. Tämä osio ei eroa merkittävästi muista videopeliprojekteista.

Tämän lisäksi pelisimulaation toiminta tarvitsee dataa, joka määrittelee esimerkiksi mitä yksiköitä ja hyökkäyksiä pelissä on, mitä eri hyökkäysten efektit tekevät ja mitkä ääniefektit kuuluvat kun jokin yksikkö tuhoutuu. Tämä data yleensä ladataan palvelimelta. Simulaatio-ohjelman tehtävänä on tulkita tämä data toimivaksi pelijärjestelmäksi.

Eri pelisimulaatioon liittyvien arvojen määritteleminen erillisen datan kautta ei sinänsä ole mitään uutta peliprojekteissa. On kohtuullisen helppoa asettaa numeroarvoja taulukkoon ja tämän jälkeen siirtää ne peliä simuloiville tietokoneille.

Battle Legionissa käytetyn järjestelmän tekee kiinnostavaksi se, kuinka paljon pelisimulaation toiminnasta määritellään palvelimelta ladattavien arvojen kautta. Esimerkiksi uusia hyökkäyksiä, yksiköitä ja niiden toimintaa voidaan säädellä yksinkertaisesti muokkaamalla yhtä taulukkoa – itse lähdekoodiin ei tarvitse koskea.

3.2 Järjestelmän käyttäminen

Pelisimulaation toimintaan käytettävät tiedot määritellään taulukoihin, joita pelitiimin jäsenet voivat käydä muokkaamassa. Nämä taulukot sisältävät esimerkiksi pelin kaikkien yksiköiden tiedot käyttäjälle luettavassa muodossa, ja niiden muokkaaminen on kohtuullisen helppoa.

Taulukot voidaan lähettää erilliselle palvelimelle. Palvelin tulkitsee taulukoiden tiedot, tekee niille joitain tarkistuksia ja lopulta tallentaa ne vanhojen tietojen tilalle. Monia palvelimen asetuksia voidaan muuttaa käytännöllisellä työkalulla, jonka käyttämiseen ei tarvita merkittävää teknistä osaamista. Pelin kehitystä varten on varattu useita palvelimia, jotta pelistä voi olla useita eri versioita kehitteillä samaan aikaan. Esimerkiksi pelin testaaminen ja kehittäminen tapahtuvat erillisillä palvelimilla, joilla voi olla eri asetukset ja/tai eri palvelinversio.

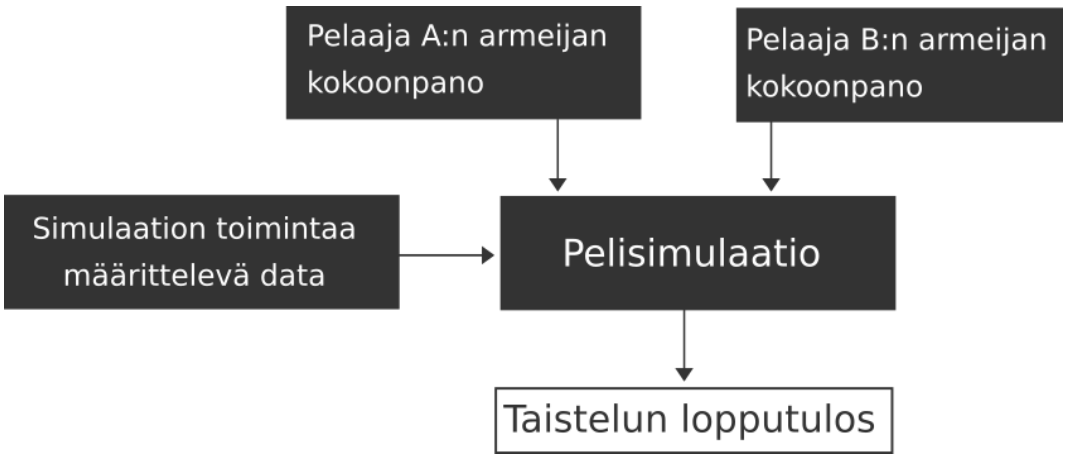
Tämän jälkeen simulointiin käytettävä data voidaan siirtää peliä simuloiville laitteille, mikä tapahtuu useimmiten ohjelman seuraavan käynnistyksen yhteydessä. Ohjelmaa ei tarvitse kääntää uudestaan, jotta muutokset tulevat voimaan. Järjestelmän avulla muutokset saadaan peliin testattavaksi noin 5-10 minuutin viiveellä, olettaen että ongelmia ei kohdata.

4 JÄRJESTELMÄN TOIMINTA

4.1 Simulaatiodatan käyttäminen

Palvelimelta ladattu pelisimulaatioon liittyvä tieto on jäsennelty tavalla, joka ei ole sellaisenaan valmis simulaation käyttöön. Tämä tieto täytyy käsitellä ja muuttaa datarakenteiksi, joita voidaan käyttää itse simulaatiossa. Tästä tiedosta käytetään jatkossa termiä simulaatiodata. Datan siirtäminen verkossa eri laitteiden välillä vaatii myös oman prosessinsa, jota ei käydä tässä opinnäytetyössä läpi.

Voidaan siis ajatella, että pelisimulaatio on rakennettu käyttämään pelilogiikan määrittelevää dataa yhtenä simulaation syötteenä. Toinen simulaation syötteistä on pelaajien armeijat, jotka pelaajat rakentavat erillisessä peliohjelman osiossa. Kuvio 1 havainnollistaa järjestelmän toimintaa.



KUVIO 1. Kaavio pelisimulaation syötteistä ja tulosteesta

Koska simulaatiodata on erillinen syöte, se mihin simulaatiodata on tallennettu ja mitä versiota simulaatiodatasta käytetään on toissijaista järjestelmän toiminnalle. Battle Legionissa simulaatiodata tallennetaan palvelimelle ja siirretään sieltä, mutta yhtä hyvin simulaatiodatasta voisi olla versioita tallessa yksittäisillä tietokoneilla.

4.2 Simulaation muuttaminen simulaatiodatalla

Simulaatiodataa voidaan käyttää eri tavoin määrittelemään pelisimulaation toimintaa. Battle Legionin pelisimulaatiossa simulaatiodataa on käytetty seuraavilla tavoilla:

1. Simulaatiodatassa olevan arvon käyttäminen funktion syötteenä. Tämä on yksinkertaisin tapa käyttää simulaatiodataa. Sen sijaan, että funktiossa käytettäisiin ohjelmaan valmiiksi määriteltyä arvoa, korvataan arvo simulaatiodatasta saadulla arvolla. Esimerkiksi yksiköiden hyökkäyksien vahinkoarvot määritellään simulaatiodatassa.
2. Simulaatiodatan käyttäminen ehtolauseen ehtona. Tämä on yksinkertaisin tapa haarauttaa simulaatio-ohjelman suoritusta. Esimerkiksi yksiköiden kykyihin voi simulaatiodatan avulla määritellä, pysähtyykö yksikön liikkuminen hyökkäyksen ajaksi vai ei. Tämä tieto on tallennettu boolean-arvona, jota käytetään haarauttamaan hyökkäysten aloittamiseen liittyvän koodin suorittamista.
3. Simulaatiodatasta hankitun arvon käyttäminen taulukon avaimena. Kyseessä voi olla suora numeroarvo, jota käytetään taulukon indeksinä. Luettua arvoa voidaan käyttää hajautustaulun (eng. ”hashtable”) avaimena. Tämä on käytännöllistä silloin, kun simulaatiodatassa halutaan viitata useaan otteeseen samaan dataan. Data voidaan tallentaa taulukkoon ja tämän jälkeen viittaukset dataan tallennetaan numerona tai merkkijonona (eng. ”string”). Simulaatiodatasta hankittua avainta voi siis ajatella muistiosoitteena. Esimerkiksi hyökkäykset, jotka laukaisevat ilmaan ammuksen sisältävät tiedon siitä, mikä ammus ilmaan ammutaan. Eri pelissä olevat ammukset on määritelty erillisessä taulukossa, ja näihin ammuksiin viitataan merkkijonon muodossa olevalla avaimella.
4. Taulukoihin voi myös tallentaa dataa käyttäen simulaatiodatasta määriteltyä avainta. Esimerkiksi yksiköillä on erilaisia kykyjä, jotka tallennetaan pelisimulaation aikana eri taulukoihin riippuen siitä, milloin kyvyn halutaan aktivoituvan. Näin eri kyvyille voidaan määrittää, aktivoituuko se taistelun alussa, hyökkäyksen aikana, vai yksikön kuollessa. Näin taulukko, mihin kykyyn tallennetaan on itsessään informaatiota, jota käytetään simulaation aikana (Fabian 2018).

5. Tyyppitiedon käyttäminen ohjelman suorituksen aikana. C#-ohjelmointikielessä on reflection-nimellä kulkeva ominaisuus, jonka avulla ohjelman suorituksen aikana päästään käsiksi ohjelmassa olevien datatyyppien tietoihin (Microsoft 2017: Reflection in the .NET Framework). Esimerkiksi luokkia voidaan ladata muistiin dynaamisesti ilman, että ladattavan luokan tyyppiä tarvitsee määritellä kääntämisvaiheessa. Näin ohjelman suoritusta voidaan haarauttaa virtuaalifunktioiden ja polymorphismin keinoin.

Tiivistettynä palvelimelta saatu simulaatiodata joko muutetaan arvoiksi, jotka tallennetaan pelin muistiin (esimerkiksi yksikön iskupisteet tallennetaan lukuna taulukkoon, joka sisältää kaikkien yksiköiden tiedot), tai pelidataa käytetään vaikuttamaan pelisimulaation suoritukseen ohjelman haarautumispisteissä.

4.3 Esimerkki järjestelmän toiminnasta

Battle Legionissa on yksikkö nimeltään Brute. Tämän yksikkö hyökkää suurella vasaralla, jonka isku aiheuttaa vahinkoa ja heittää vihollisyksiköitä ilmaan (Kuva 4). Kuten kaikki pelin kyvyt, on tämä hyökkäys määritelty simulaatiodatassa.

KUVA 4. Ruutukaappaus Brute-yksikön hyökkäyksestä.

Taulukossa on rivi ”Abilities” eli kyvyt, joka sisältää eri yksiköiden kyvyt, jotka voivat aktivoitua pelisimulaation aikana. Kyvyt ovat käytännössä kokoelma efektejä, joille on määritelty tietty kohta pelilogiikassa jolloin ne aktivoituvat. Esimerkiksi jotkin hyökkäykset aktivoituvat, kun vihollisyksikkö on kantaman päässä tai kun kyvyn omistama yksikkö kuolee. Bruten hyökkäys aktivoituu, kun viholliskohde on 5 mittayksikön päässä (mittayksiköt eivät ole verrannollisia olemassa oleviin mittayksiköihin). Taulukot 1, 2, 3 ja 4 esittävät, kuinka Bruten hyökkäys on määritelty simulaatiodatassa.

TAULUKKO 1. Bruten hyökkäyskyvyn tiedot simulaatiodatassa.

abilityID	BruteAttack
abilityRange	5.0
abilitySpeed	1.0
abilityCooldown	1.0
animationName	Attack
Effects	BruteConeEffect,BruteKnockback,Damage
affectMoving	true
audioAbilitySingle	Unit_AttackSingle_Brute
aliveThreshold	0
requireTarget	TRUE
singleUse	FALSE

Yksittäinen efekti on käytännössä pienin yksittäinen tapahtuma, joka vaikuttaa pelimaailmaan. Yhdistämällä useita efektejä yhteen saadaan luotua monimutkaisia tapahtumia yksinkertaisista rakennuspalikoista. Bruten hyökkäys sisältää seuraavanlaiset efektit:

1. Efekti, joka määrittelee millä kriteereillä yksiköt lasketaan hyökkäyksen muiden efektien vaikutuksen alaiseksi.
2. Puskuefekti, joka puskee kohteena olevia yksiköitä poispäin hyökkäävästä Brutesta ja tämän lisäksi lennättää ilmaan kevyempiä yksiköitä. Käytännössä yksiköiden simulaatiotilaan kuuluu massa ja nykyinen kiihtyvyys, ja tämä efekti lisää yksiköihin kiihtyvyyttä poispäin Brutesta.

3. Vahinkoefekti, joka yksinkertaisesti vähentää kohteena olevien yksiköiden iskupisteitä Bruten hyökkäysarvon verran. Kun yksiköiden iskupisteet ovat nolla tai vähemmän, yksikkö kuolee.

TAULUKKO 2. Bruten hyökkäyksen kohteiden valinnan määrittelevän efektin sisältö simulaatiodatataulukossa

effectID	BruteConeEffect
effectType	TargetConeEffect
maxResults	100
range	7.0
arc	125.0
targetMask	EnemyAllGround,DeadGround,EnemyStructure
particleEffect	Smash

TAULUKKO 3. Bruten hyökkäyksen puskuefektin sisältö simulaatiodatataulukossa.

effectID	BruteKnockback
effectType	Knockback
force	275.0

TAULUKKO 4. Bruten hyökkäyksen vahinkoa aiheuttavan efektin sisältö simulaatiodatataulukossa

effectID	Damage
effectType	CauseDamage

Efekteihin jaoteltuna voidaan huomata, että Brute-yksikön hyökkäys koostuu selkeistä palikoista, joita pystytään siirtämään, uudelleenkäyttämään ja muovaamaan erikseen. Jos Bruten hyökkäyksen haluttaisiin jäädyttävän vihollisia tai käyttävän erilaista logiikkaa kohteiden määrittelemisessä, ei se vaatisi kuin muutaman solun muuttamista taulukosta. Näin ollen koko hyökkäyksen logiikka voidaan muokata taulukon avulla. Tämän lisäksi hyökkäyksen toimintaa voidaan hienosäätää muuttamalla efekteihin liittyviä arvoja.

Eri efektit ja niiden logiikka täytyy ohjelmoida osaksi pelin lähdekoodia. Kun tämä työ on tehty, efektejä voidaan käyttää vapaasti osana pelin eri yksiköiden toimintaa. Efektit käytännössä määrittelevät, mihin yksiköihin kyky vaikuttaa, ja miten kyky vaikuttaa näihin yksiköihin. Koska yksiköt ovat pelisimulaation tärkein toistuva elementti, pohjautuvat kaikki kyvyt yksiköiden tilojen muokkaamiseen. Kaikki yksiköt sisältävät pääosin samat tiedot. Näin ollen kaikki pelin kyvyt pystyvät automaattisesti vaikuttamaan kaikkiin pelin yksiköihin. Tämän avulla pelin simulaatio pystyy tuottamaan suuren määrän erilaisia interaktioita, sillä kaikki toiminta lopulta vaikuttaa samaan simulaatiotilaan.

4.4 Esimerkki simulaatiodatan muuttamisesta

Simulaatiodataa muokkaamalla voidaan muuttaa pelissä olevien yksiköiden hyökkäyksiä. Järjestelmän toiminnan havainnollistamiseksi otamme
Brute-yksikön hyökkäyksen, ja muokkaamme sen toiminnallisuutta. Muutamme simulaatiodataa niin, että Bruten hyökkäys:

1. Ottaa kohteekseen kaikki vihollisyksiköt 10 mittayksikön kokoisen ympyrän säteellä, joka on keskitetty Brute-yksikköön.
2. Hyökkäys sytyttää kaikki kohteeksi otetut yksiköt tuleen.
3. Hyökkäys aktivoituu paljon nopeammin ja useammin kuin Bruten alkuperäinen hyökkäys.
4. Hyökkäys laukaisee räjähdys-partikkeliefektin hyökkäyksen keskipisteeseen.

Taulukot 5, 6 ja 7 esittävät Bruten muokatun hyökkäyksen tiedot simulaatiodatassa. Kuva 5 esittää miltä uusi Bruten hyökkäys näyttää pelisimulaation aikana.

TAULUKKO 5. Bruten uuden hyökkäyksen tiedot simulaatiodatassa. Muokatut solut on korostettu.

abilityID	BruteAttack
abilityRange	5.0
abilitySpeed	1.0
abilityCooldown	**0.8**
animationName	Attack
Effects	**BruteConeEffect,ApplyBurn**
affectMoving	true
audioAbilitySingle	Unit_AttackSingle_Brute
aliveThreshold	0
requireTarget	TRUE
singleUse	FALSE

TAULUKKO 6. Bruten uuden hyökkäyksen kohteiden valinnan määrittelevä efekti.

effectID	BruteConeEffect
effectType	**TargetCircleEffect**
range	**10.0**
targetMask	EnemyAllGround,DeadGround,EnemyStructure
particleEffect	**BombExplosion**

TAULUKKO 7. Bruten uuden hyökkäyksen sytytysefektin tiedot.

effectID	ApplyBurn
effectType	ApplyBuff
buffname	Burn

Tulessa oleminen on yksi erilaisista tiloista, missä yksiköt voivat olla. Näistä tiloista käytetään termiä ”Buff” simulaatiodatataulukossa. Taulukko 8 sisältää simulaatiodatassa olevat tiedot, jotka liittyvät tulessa olemiseen.

TAULUKKO 8. Tulessa olemisen tiedot.

buffID	Burn
ticks	5
tickInterval	0.5
maxStacks	1
tags	Burn, Condition
onApplyEffects	UpdateBurnVisuals
onTickEffects	BurnHealthDamage

KUVA 5. Ruutukaappaus Brute-yksikön muokatusta hyökkäyksestä.

Hyökkäyksen muuttaminen onnistui vaivattomasti vain tekemällä muutoksia simulaatiodatataulukkoon. Tarvittiin vain muutama minuutti, jotta uusi toiminnallisuus saatiin näkyviin pelissä.

4.5 Järjestelmän rajoitteet

Koska simulaatiodata ei lisää toiminnallisuutta käännettyyn simulaatio-ohjelmaan, on kaikki simulaatiossa mahdollisesti tapahtuva toiminta ensin ohjelmoitava osaksi simulaatiota. Simulaatiodatan avulla ei voi luoda uusia funktioita. Tämä tarkoittaa, että jos simulaation toimintaa halutaan laajentaa, se vaatii ohjelmointimuutoksia. Esimerkiksi uusien yksiköiden luominen peliin vaatii useimmiten simulaatio-ohjelman muokkaamista ja kääntämistä. Käytännössä simulaatio-ohjelma määrittelee mitä kaikkea simulaatiosta pystytään simulaatiodatan avulla muokkaamaan.

5 JÄRJESTELMÄN HYÖDYT

5.1 Simulaation muuttamisen helppous

Simulaatiodataa muokkaamalla pelisimulaatioon voi tehdä merkittäviä muutoksia ilman, että muutosten tekijän täytyy osata ohjelmoida tai että hän ymmärtäisi kuinka pelisimulaatio on ohjelmoitu. Tämän lisäksi simulaatiodatan sisältö on jäsennelty luettavaan muotoon. Tämän avulla on helppo löytää eri yksiköiden kyvyt ja peliin liittyvät arvot.

Useimmiten simulaatiodataan halutaan tehdä yksittäisiä muutoksia, ja muutoksen tekijällä on selkeä kuva siitä mitä muutoksella halutaan saada aikaan simulaatiossa. Tällaisen yksittäisen muutoksen tekeminen simulaatiodatataulukkoon vaatii paljon vähemmän tietoa simulaation toiminnasta kuin samanlaisen muutoksen tekeminen simulaatio-ohjelmaan.

5.2 Simulaation toiminnan iteroimisen nopeuttaminen

Pelisuunnittelijan täytyy usein tehdä paljon pieniä muutoksia pelisimulaation toimintaan, jotta pelisimulaatio saadaan vastaamaan pelisuunnittelijan päämääriä. Simulaatiodataa muokkaamalla monet muutokset voidaan saada aikaan ilman ohjelmointityötä, minkä takia pelisuunnittelija pystyy keskittymään pelin toiminnan iteroimiseen. Tämä vähentää iteroimiseen menevää aikaa merkittävästi.

6 KOHDATUT ONGELMAT TOTEUTUKSESSA

6.1 Alttius käyttäjien virheisiin

Pelisimulaatioon käytettävä tieto määritellään ensimmäiseksi taulukossa, jota eri pelinkehitystiimin jäsenet muokkaavat. Tämä muokkausvaihe on kuitenkin altis inhimillisille virheille. Taulukon eri solut sisältävät usein tietoa mikä tulkitaan eri tavoin riippuen tiedon käyttötarkoituksesta. Esimerkiksi solut voivat sisältää lukuja, merkkijonoja, numeroita ja yksittäisiä elementtejä määritellystä kokoelmasta.

Tarkistusta siitä, onko käyttäjän kirjoittama tieto oikein tulkittavissa ei tapahdu sisällön kirjoittamisvaiheessa. Kirjoitussäännöt voivat olla usein hyvin pikkutarkkoja, esimerkiksi pisteet, pilkut ja lainausmerkit aiheuttavat usein ongelmia. Tämän lisäksi pelilogiikassa usein viitataan asioihin uniikeilla nimillä, jotka pitää kirjoittaa taulukkoon oikein. Kirjoitusvirheen aiheuttamat ongelmat voivat olla vaikeita löytää. Taulukon tarkistaminen virheiden osalta on hidasta työtä, mikä vie aikaa ja on itsessään altis virheille. Kuten Barchard, Pace. ja Burns totesivat tutkimuksessaan (Barchard K., Pace L. ja Burns S. 2009), on käsin kirjoittaminen aina altis virheille, vaikka työ tarkistettaisiin kertaalleen.

6.2 Virheiden huomaamiseen menevä aika

Koska simulaatiodata on syöte, jonka pelisimulaatio ottaa käsiteltäväkseen, on sitä ajateltava syötteenä myös ohjelman virheiden puolesta. Simulaatiodata ei ole käyttäjän määrittelemä syöte, joten tarkistamista voidaan tehdä ennen kuin syöte menee simulaation käsiteltäväksi. Kuitenkaan simulaation kääntämisvaiheessa itse simulaatiodatasta ei tiedetä mitään, mikä tarkoittaa että simulaatiodatassa olevat virheet tapahtuvat simulaation suorituksen aikana.

Varsinkin jos virheellistä tietoa ei tunnisteta simulaation alustusvaiheessa, voi virheiden tunnistamiseen mennä pitkä aika, tai virheet voivat jopa jäädä huomaamatta. Yleisenä periaatteena ajatellaan, että virheiden tunnistamisen on parasta tapahtua kääntämisvaiheessa, sillä suoritusvaiheessa virheitä on yhä vaikeampi diagnosoida ja korjata (Difference Between Compile Time and Runtime: 2018).

Simulaatiodatassa olevat virheet voidaan siis tunnistaa seuraavissa eri vaiheissa. Vaiheet on listattu sen mukaan kuinka pitkään vaiheeseen pääseminen kestää, ja näin ollen kuinka vaikeaa virheen todentaminen on. Järjestys on nouseva sen mukaan, kuinka pitkään virheen löytämiseen keskimäärin menee.

1. Simulaatiodatan sisältävän muokattavan taulukon tallennusvaihe. Tässä kohtaa simulaatiodatataulukko muutetaan JSON-tiedostomuotoon (Ecma International 2017). Jos JSON-tiedoston kirjoittaminen epäonnistuu, tallennusvaihe epäonnistuu ja ongelmasta tulostetaan viesti käyttäjälle.
2. Simulaatiodataa käsitellään ensi kerran pelisimulaatiota suorittavassa tietokoneessa. Tämä tapahtuu, kun simulaatiota suorittava tietokone käynnistetään uuden simulaatiodatan kanssa.
3. Kun yksikkö/hyökkäys aktivoituu pelisimulaatiossa ensimmäistä kertaa. Jos on tiedossa mitä yksiköitä koskeva simulaatiodata on muuttunut, on selkeää testata nämä yksiköt pelissä kerran. Pelissä on kuitenkin melko monta yksikköä, joten kaikkien yksiköiden rutiininomainen tarkastaminen on vaikeaa.
4. Kun jokin erityinen interaktio tapahtuu pelisimulaation sisällä. Koska peli sisältää todella paljon mahdollisia tapahtumia ja interaktioita, on erityisen ongelmallista jos jokin virhe tulee näkyviin vain erityistapauksissa. Tällöin virheen tunnistaminen voi kestää todella pitkään. Voi myös olla epäselvää onko kyseessä edes virhetila, vai onko kyseessä haluttu toiminta. Mitä pidempi aika muutoksen tekemisen ja virheen tunnistamisen välillä on, sitä vaikeampaa on saada selville, mikä muutos aiheutti virhetilan.

6.3 Sisäisen dokumentaation puute

Vaikka pelisimulaation tiedon sisältävään taulukkoon kirjoitettu tieto olisi täysin oikein teknisestä näkökulmasta, on silti mahdollista, että käyttäjä tulkitsi taulukon solun vaikutuksen pelisimulaatioon väärin. Esimerkiksi joissain efekteissä on kenttä: ”DamageReductionRate”. Tämä kenttä on liukuluku, mutta sen skaala ei tule ilmi sen nimestä eikä kentän toiminnasta ole kirjoitettu muualla. Käyttäjä voi esimerkiksi kirjoittaa ”-0.5” olettaen, että tämä vähentää vahinkoa 50%. Kenttä kuitenkin toimii niin, että positiiviset luvut merkitsevät kuinka paljon vahinkoa vähennetään, joten ”-0.5” lisääkin vahingon määrää vähentämisen sijaan.

Pelin kehitykseen käytettävien työkalujen toiminta usein siirtyy suullisesti pelinkehittäjältä toiselle, mikä voi joskus aiheuttaa ongelmia ja hämmennystä. Pelin ja pelinkehitykseen käytettävät työkalut myös muuttuvat aktiivisen kehityksen myötä, ja eri muutosten perässä pysyminen voi olla hankalaa. Työkalujen on tarkoitus olla sellaisten kehittäjien käytössä, jotka eivät ole osana pelin ohjelmiston kirjoittamista (esimerkiksi pelisuunnittelijat ja QA-testaajat). Tiedon siirtyminen on siis ongelma.

6.4 Lisääntynyt koodin haarautuminen

Koska simulaatiodata on syöte, jota pelisimulaatio käyttää, on pelisimulaatio ohjelmoitava niin, että simulaatiodatan sisältöä ei tiedetä etukäteen. Tämä tarkoittaa, että simulaation toimintaa on vaikea arvioida ilman, että koodia käännetään ja ajetaan simulaatiodatan kanssa. Voi esimerkiksi olla, että joitain funktioita ei koskaan käytetä johtuen simulaatiodatan sisällöstä. Tämä lisää simulaatio-ohjelman monimutkaisuutta. Simulaatiodata ei muutu versiosta toiseen sunnattoman paljon, joten simulaatiodatan sisällöstä voi tehdä melko tarkkoja arvioita, jotka pysyvät ajankohtaisina pitkään.

7 POHDINTA

Mitä enemmän tutkin Battle Legionissa käytettyä järjestelmää, jossa pelisimulaatiota voidaan määritellä palvelimelta tulevan datan avulla huomasin, että koko järjestelmä on lopulta hyvin yksinkertainen. On kohtuullisen yksiselitteistä, että dataa kuten numeroarvoja voidaan ladata palvelimelta ja tämän jälkeen käyttää syötteenä peleissä. Battle Legionin järjestelmä vain hieman laajentaa tätä konseptia, sillä dataa käytetään laajamittaisesti myös haarauttamaan pelin simulaation koodin suoritusta.

Tämän tajuttuani järjestelmä tuntui paljon vähemmän kiinnostavalta aiheelta kuin aloittaessani ajattelin. Perusperiaatteet, jotka ajavat järjestelmää, ovat hyvin yksinkertaiset. Monet toteutukseen liittyvät yksityiskohdat eivät mielestäni olleet syvemmän selittämisen ja tutkimisen arvoisia. Tämä johti siihen että leikkasin useita suunniteltuja osioita pois tästä opinnäytetyöstä. En kuitenkaan kadu sitä, sillä mielestäni on tärkeämpää sanoa asiansa tiiviisti, kuin yrittää luoda sisältöä siitä mistä sitä ei saa.

Tutkittua järjestelmää, on mielestäni sitä paras ajatella kehitystyökaluna, jonka avulla pelisimulaation toimintaa voi muovata ilman lähdekoodin muokkaamista ja uudelleenkääntämistä. Tähän tarkoitukseen järjestelmä toimii mielestäni hyvin.

Opinnäytetyö on mielestäni kohtuullisen onnistunut sille asettamissani tavoitteissa.

LÄHTEET

Barchard K, Pace L ja Burns S 2009. Double Entry: Accurate Results from Accurate Data. Luettu 5. heinäkuuta 2019. http://barchard.faculty.unlv.edu/doubleentry/Double%20Entry%20APS%202009%20handout.pdf

Difference Between Compile Time and Runtime 2018. Luettu 3. heinäkuuta 2019. www.differencebetween.com/difference-between-compile-time-and-vs-runtime/

ECMA International 2017. The JSON Data Interchange Syntax. Luettu 4. heinäkuuta 2019. www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Fabian R 2018. Data-Oriented Design. Luettu 30. kesäkuuta 2019. www.data-orienteddesign.com/dodbook/

Microsoft .NET Documentation. Reflection in the .NET Framework 2017. Luettu 3. heinäkuuta 2019. docs.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection