Henry Flink

Selainpelin taistelumoottori

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Opinnäytetyö
7.6.2011

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Henry Flink Selainpelin taistelumoottori 35 sivua + 5 liitettä 7.6.2011
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Simo Silander, Lehtori Miikka Mäki-Uuro, Lehtori

Insinöörityössä lähdettiin tekemään taistelumoottoria alun perin 2000-luvun alussa toimineeseen peliin, josta ruvettiin myöhemmin tekemään samanlaista uutta peliä. Taistelumoottori on se osa pelistä, joka ratkaisee kahden osapuolen välisen taistelun tuloksen, kun sille antaa syötteenä niiden tiedot ja alusmäärät. Se on siis pohjimmiltaan PHP-ohjelmointikielellä kirjoitettu algoritmi.

Työssä puhutaan ensin lähtökohdista, joissa hahmottuu tarkemmin peli, jota varten taistelumoottoria tarvittiin. Samalla käsitellään, mitä täytyy huomioida palvelinalustalta, jonka päällä peli pyörii. Etenkin PHP:n rajoitteet käydään tarkasti läpi. Pohjustuksesta siirrytään tutkimaan miten kaikki taistelumoottorin osa-alueet toimivat ja millä tapaa ne rakentavat taistelumoottorin rakenteen.

Testaus oli tärkeä osa taistelumoottorin kehitystä, sillä se on erittäin keskeinen osa peliä, johon se tehtiin ja sen pitää toimia odotetulla tavalla. Testauksessa keskityttiin kolmeen osa-alueeseen: kuormituskestävyyteen ja suorituskykyyn sekä tulosten oikeellisuuteen. Testauksen apuna käytettiin itse kirjoitettuja apuohjelmia, erilaisia testitapauksia ja taistelumoottoria käyttävää simulaattorisivua. Myös yhteisö pääsi mukaan testaukseen simulaattorisivun avulla.

Taistelumoottori saatiin valmiiksi ja testauksessa vakaaksi. Sen suorituskyky oli hyvä ja se on ollut vakaassa käytössä pelissä siitä alkaen. Työn ohessa tehtiin myös monia huomioita asioista, jotka olisi voinut tehdä toisin, jos ne olisi tajunnut heti projektin alkaessa. Näitä asioita on pohdittu ja niihin on myös esitetty ratkaisuja.

Avainsanat	PHP, taistelumoottori, peli

Metropolia

Abstract

Author(s) Title Number of Pages Date	Henry Flink Battle engine for a browser game 35 pages + 5 appendices 7 June 2011
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Miikka Mäki-Uuro, Senior lecturer

This thesis is about a battle engine for a current remake of a game named EmpireQuest that first ran in the early 2000's. A battle engine is the part of the game that solves battles between two sides when you give it input of the participants ships and data. So in essence it is an algorithm written with PHP programming language.

The thesis starts by telling the story behind the game and then continues to explain the platform the game is running on. Special notes were made about the restrictions that using PHP as a programming language has. The study continues by telling how the separate parts of the battle and its mechanics work and then explains how the parts construct the structure of the battle engine.

Testing was an important part of the development of the battle engine, because it is such an essential part of the game and it is always assumed to produce correct results. In testing the attention concentrated on three parts: load testing and performance as well as correctness of the calculations. As tools there were purpose built PHP programs, test cases and a battle simulator web page. There was also some testing done by the community of gamers with the simulator web page.

The battle engine was finished and testing was successful. Its performance proved to be good and it was stable when integrated to the game it was made for. During the making of the battle engine, there were observations made about things that could have been done better, if only they had been known before hand. As hindsight there is some contemplating and solutions to these observations towards the end of the study.

Keywords	PHP, battle engine, game

Sisällys

Liitteet

Liite 1. EmpireQuestin-alusten käyttäytyminen taisteluissa

Liite 2. Kuvia EmpireQuest-pelistä

Liite 3. Syöte- ja palautetaulukon ennalta määrätty sisältö

Liite 4. Suorituskykymittauksen testauspöytäkirja

Liite 5. Erilaisia taistelumoottorin testitapauksia

Lyhenteitä ja käsitteitä

Algoritmi	Ohjelma, joka ratkaisee ongelman tai tehtävän.
EmpireQuest	Avaruuteen sijoittuva selaimella pelattava strategiamoninpeli.
Funktio	Ohjelmalohko joka voidaan suorittaa kutsumalla sen nimeä tarvittavine parametreineen.
Internet-selain	Kun tekstissä puhutaan selaimesta, tarkoitetaan sillä Internet-selainta kuten Internet Exploreria, Firefoxia, Operaa tai Chromea.
MySQL	SQL-tietokannan hallintajärjestelmä.
Palvelin	Tietokone, jonka pääasiallinen tehtävä on toimittaa palveluita usealle käyttäjälle.
Parametri	Funktiolle toimitettava tieto, joka voi olla mikä vain sallittu muuttuja kuten esimerkiksi merkkijono, kokonaisluku tai taulukko.
PHP	Lyhenne sanoista PHP: Hypertext Preprocessor on Perlin kaltainen ohjelmointikieli, jota käytetään erityisesti Web-palvelinympäristöissä dynaamisten web-sivujen luonnissa.
Selainpeli	Selaimella Internetin yli pelattava tietokonepeli ilman, että omalla koneelle tarvitsee asentaa mitään.
Web-palvelin	Palvelimella sijaitseva palvelu, joka toimittaa pyytävälle selaimelle sen haluaman materiaalin eli verkkosivun.

1 Johdanto

Työssä lähdettiin tekemään EmpireQuest-nimiseen [1] selainpeliin taistelumoottoria. Pelimaailma sijoittuu avaruuteen, jossa pelaajalla on mahdollisuudet hallita maksimissaan 6 planeettaa, joissa on jokaisessa 4 laivastoa. Yksi on kotilaivasto, joka ei voi liikkua ja sisältää myös planeetan puolustusrakennukset, sen lisäksi on kolme laivastoa, jotka voivat liikkua ympäri pelin universumia. Pelissä on myös poliittinen puolensa, joka pelissä on kuitenkin taistelussa, jolla vastustajalta voidaan tuhota aluksia, varastaa resursseja ja valloittaa planeettoja. Taistelumoottorin osuus on ratkaista taistelujen tulokset hyökkääjän ja puolustajan välillä. Uusi taistelumoottori tarvittiin, koska edellinen jo pelissä ollut taistelumoottori oli epävakaa ja sen laskemat tulokset olivat kelvottomia.

Taistelumoottorin tekeminen alkoi suunnittelusta ja etenkin sen noudattamien sääntöjen kirjaamisesta. Koska kyseessä on verkkosovellus, oli myös tarpeellista kartoittaa käytössä olevan sovellusalustan rajoitteet. Lähtökohtana oli pelin manuaali [2], jossa säännöt oli selitetty. Pelissä on monenlaisia aluksia, jotka käyttäytyvät eri tavoin. Kaikilla aluksilla on omat ominaisuutensa ja tyyppinsä, jotka määrittävät niiden toimintatarkoituksen. Alus voi esimerkiksi ampua normaalisti, jäädyttää tai vaikka varastaa resursseja. Lisäksi aluksilla on vielä toimintajärjestys, jolla ne toimivat. Tämä mahdollistaa muun muassa sen, että hitaampia aluksia voi tuhota nopeampien alusten toimesta, ennen kuin ne ehtivät toimia. Nämä eri käyttäytymismallit lisäävät peliin monimuotoisuutta, mutta samalla myös vaikeuttavat taistelumoottorin suunnittelua, ohjelmointia ja testausta.

Viimeisenä vaiheena oli testaus, jota lähestyttiin monesta eri näkökulmasta, koska taistelumoottorilla on niin monia eri käyttötapauksia. Suljetussa käyttäjäpiirissä käytiin läpi ensin ennalta määriteltyjä testitapauksia, joita varten tehtiin myös testausta helpottava erillinen selaimella käytettävä simulaattorisivu [3]. Taistelumoottoriin lisättiin myös erillisiä tulostuksia, jotka tulostivat taistelun eri vaiheet askel askeleelta ruudulle. Tällä tavoin saatiin nopeasti syötettyä erilaisia taisteluita ja tarkistettua että alukset tekivät, mitä niiden kuuluikin ja että lopputulos oli oikein. Tällä tavoin testaaminen on erittäin työlästä, koska tulokset pitää erikseen laskea etukäteen käsin.

Selainpelin osana ei kumminkaan riittänyt että taistelumoottori toimi halutusti, vaan piti myös varmistaa, että se toimii tehokkaasti ja vakaasti. Taistelumoottoria kuormitettiin eri alusmäärillä ja peräkkäisillä toistokerroilla, jotta saatiin selville satunnaiset virheet, jotka voisivat keskeyttää taistelumoottorin suorituksen. Samalla saatiin selville se, miten paljon taisteluita kyettäisiin suorittamaan olemassa olevan ajan puitteissa ja vertaamaan tulosta siihen, mikä pelin kannalta on hyväksyttävää.

2 Lähtökohdat

Taistelumoottorin idean ja toiminnan ymmärtämiseen vaaditaan hieman tuntemusta lähtökohdista ja etenkin pelistä, jonka päälle taistelumoottoria alettiin ohjelmoida. Tässä luvussa keskitytään selittämään niitä lähtökohtia, joihin sisältyy myös palvelinalusta, jonka päällä peli ja taistelumoottori toimivat.

2.1 EmpireQuest-peli

EmpireQuest oli yksi ensimmäisiä avaruuteen sijoittuvia ilmaisia selainpelejä 2000-luvun alussa, jolloin selainpelit olivat saamassa alkusysäyksiään ja näitä pelejä mahdollistavat tekniikat ja alustat olivat vielä uusia. Sittemmin pelit ovat kehittyneet huomattavasti ominaisuuksiltaan sekä grafiikaltaan ja niitä on tullut ja mennyt lukemattomia määriä. Monet näistä ovat ottaneet vanhojen pelien ideat ja vieneet ne uudelle tasolle tehden niiden pyörittämisestä ison luokan liiketoimintaa.

EmpireQuest ja sen kultaiset ajat sijoittuivat 2000-luvun alkuun. Se oli harrastelijaprojekti, joka myös näkyi pelin vakaudessa ja ylläpidossa. Oli uusi asia, että peliä pystyi pelaamaan Internetissä. Vaikka moninpelejä oli olemassa, ne vaativat pelin asennettuna paikallisesti omalle koneelle sekä lähiverkon, jonka kautta sitä pystyi pelaamaan muiden kanssa. Selainpelien myötä saman pystyi tekemään pelkällä Internet-selaimella kotoa käsin. Pelissä oli myös muita ennalta näkemättömiä asioita: erilaisia aluksia ja ampumistyyppejä, liittoutumia sekä aktiivinen yhteisö. Hyvät puolet saivat aikaiseksi uskollisen käyttäjäkunnan, jonka lukumäärä oli kymmenissä tuhansissa. Monen selainpelin menetys riippui juuri nimenomaan yhteisöstä ja siitä, kuinka ystävällinen se oli

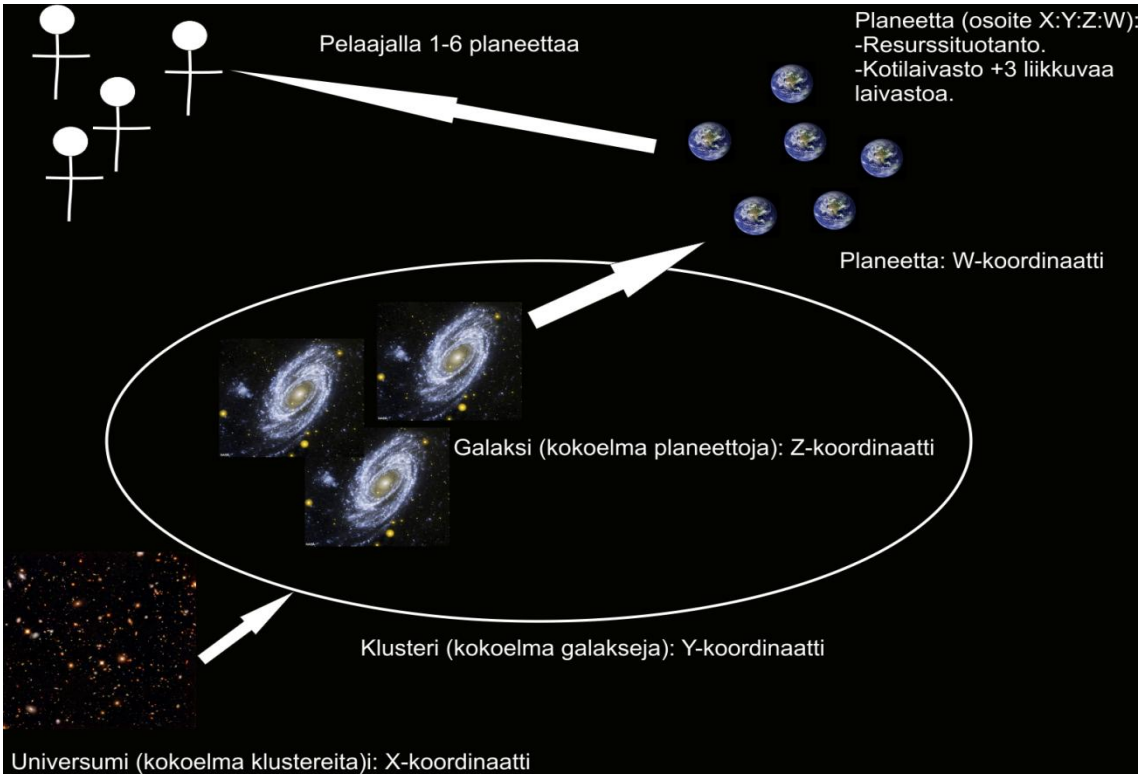
uusillekin pelaajille. Pelistä itsestään saattoi monessa tapauksessa jopa tulla toissijainen asia, koska ihmisten välinen kanssakäyminen oli tärkeämpää. Näitä pelejä voisi jopa kuvata ensimmäisiksi sosiaalisiksi medioiksi.

EmpireQuestissa oli kuitenkin ongelmia, jotka lopulta johtivat sen sulkeutumiseen. Etenkin taistelujen kanssa oli alusten määrien noustessa ongelmia, jotka johtivat useimmissa tapauksissa siihen, että peli ja palvelin kaatuivat kuorman alla. Tämä aiheutti käyttäjäkunnassa eripuraa, koska se saattoi aiheuttaa epäreilua hyötyä toisille. Suunnitteilla oli uusi versio pelistä, jossa olemassa olevan pelin toimintaa kehitettäisiin ja korjattaisiin. Lisäksi tulisi uusia ominaisuuksia, jotka olivat yhteisön ideoiden ja palautteen kautta syntyneet. Uusi versio jäi kuitenkin vain idean tasolle, koska pelin ylläpito loppui ongelmien ja kehittäjien mielenkiinnon suuntautuessa heidän pelin ulkopuolisen elämän asioihin. EmpireQuest sulkeutui ja pelaajat hajaantuivat menettäen suurimmaksi osaksi yhteyden toisiinsa.

Edellä kuvatun kaltaisille projekteille ei ollut vielä olemassa minkäänlaista pohjarakennetta, niitä pyöritettiin yksityishenkilöiden voimin normaalin arjen yhteydessä. Palvelinteknologia ja -rauta ovat sittemmin kehittyneet huomattavasti, on todennäköisestä, että peli toimisi nykyään paremmin, mutta se ei tarkoita, että samat virheet kannattaisi toistaa. Lisääntyneet resurssit eivät ole tekosyy tehdä huolimattomia ohjelmia.

Peli kumminkin jäi elämään muutaman vanhan pelaajan muistoihin, osa heistä rupesi kehittämään kopiota pelistä vapaa-ajallaan [1]. Tiimi on hyvin kansainvälinen ja siihen kuuluu ihmisiä jokaiselta maapallon mantereelta. Erityispiirteenä pitää mainita myös se, että suurin osa ryhmästä ei tunne toistensa niin kutsutun oikean elämän asioita, eli projektin henkilöillä on pienimuotoinen nimettömyys. Projektin tavoitteena on tuoda peli takaisin pelattavaksi vanhoille pelaajille, eikä niinkään luoda siitä liiketoimintaa. Kaikilla on ohessa arjen työnsä ja tehtävänsä, mistä johtuen kehitys on välillä verkkaista. Projektin edistyminen on ollut hidasta myös siksi, että mitään muuta materiaalia muistikuvien ja muutaman vanhan Internet-arkiston uumenista löytyneen ohjekirjan sivun [2] lisäksi ei ollut ja kaikki piti tehdä uudelleen alusta alkaen. Liitteessä 1 on Internet-arkistosta kopioitu Empirequestin taistelun säännöt ja kulku. Liitteessä 2 on kuvankaappauksia itse EmpireQuestin pelisivuista sekä esimerkki taistelu-uutisesta.

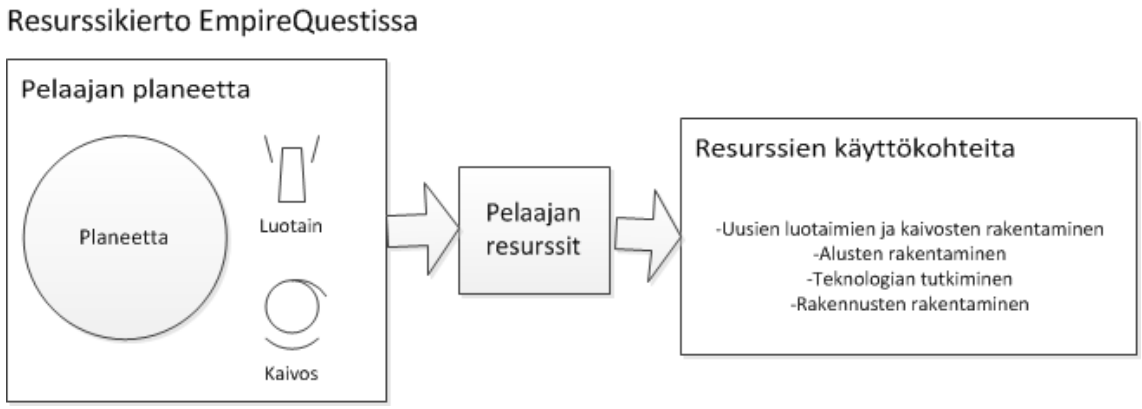
Tämä työ sai alkunsa, kun selvisi, että pelistä puuttui toimiva taistelumoottori. Sitä oli yritetty tehdä projektin kehittäjien toimesta useampaan kertaan ja lopulta peliin oli jouduttu ottamaan toisen avoimen lähdekoodin pelin taistelumoottori. Se ei kuitenkaan toiminut, vaan taistelut jäivät usein ratkaisematta, tulokset olivat täysin satunnaisia. Jopa asiat kuten alusten ominaisuuksien ja niiden vaihtelu eivät vaikuttaneet taistelujen toimintaan. Tämä oli erittäin vakava puute pelissä, koska taistelut ovat suuri osa pelin toimintaa ja sen hauskuutta.



Kuva 1 . EmpireQuestin pelimaailma

Peli pyörii planeettojen ympärillä, jossa taistelut tapahtuvat (kuva 1). Pelaaja aloittaa yhdellä kotiplaneetalla ja voi omistaa sen lisäksi maksimissaan 5 siirtokuntaplaneettaa. Kotiplaneetta on siitä erilainen siirtokuntaplaneettoihin verrattuna, ettei sitä voi valloittaa. Näillä planeetoilla on kaivoksia ja luotaimia (mines and probes), jotka tuottavat kumpikin eri resurssia (kuva 2). Näitä resurssintuottajia saa kahdella tavalla lisää: joko kuluttamalla resursseja niiden rakentamiseen tai sitten niitä voi varastaa toisien pelaajien planeetoilta. Kertyneillä resursseilla voidaan tutkia uusia teknologioita, rakentaa aluksia, rakennuksia ja resurssin tuottajia sekä vakoilla toisten pelaajien planeettoja. Uusien alusten, rakennusten ja ynnä muiden tilaaminen lisää ne tapahtumajonoon nii-

den rakennusajan määrittämään kohtaan. Rakennusaika ei siis ole määritetty kellonaikana, vaan tapahtumien (ticks) lukumääränä.



Kuva 2 . Resurssikierto EmpireQuestissa

Yksi pelin tärkeä piirre on eräajotyyppinen [4] tapahtumalaskin (ticker). Tapahtumalaskin on PHP-skripti, joka sisältää apuohjelmina kaikki tarpeelliset päivitystoimet yhden tapahtuman aikana: resurssien lisääminen pelaajille, laivojen, rakennusten ja tutkimusten edistyminen tapahtumajonossa sekä laivastojen liikkuminen. Tapahtumalaskin on siis kokoelma erilaisia apuohjelmia, joka voidaan ajaa ajoitetusti halutuin väliajoin, jota ilman pelin tapahtumajonot eivät purkautuisi tai resurssit kertyisi. Sen toimintaan eivät pelaajat pysty vaikuttamaan, vaan se ajetaan automaattisesti niiden tietojen pohjalta jotka on pelin tietokannassa. Huomattava on, että pelaajan tekemät asiat, kuten alusten tilaaminen tapahtumajonoon, tapahtuvat silloin, kun pelaaja niitä haluaa rakentaa, eli siltä osin peli on reaaliaikainen.

Tämä tarkoittaa myös sitä, että myös mahdolliset taistelut tapahtuvat vain kerran joka kerta, kun tapahtumalaskin ajetaan. Tämäntyylinen ratkaisu toimii paremmin Internet-sovelluksessa kuin se, että kaikki tapahtumat tapahtuisivat reaaliaikaisesti. Reaaliaikainen tapahtumakäsittely vaatii äärettömän paljon resursseja palvelinalustalta, jolla se pyörii. Reaaliaikaisessa toteutuksessa, palvelimen pitäisi joka sekunti tarkistaa onko uusia tapahtumia ja toimia tarvittaessa tapahtumien mukaan, sekä päivittää pelaajien resurssi- ja pistemäärät. Vaikka palvelinresursseja olisi äärettömästi, saattaisi pelaajien yhteyksien viiveet aiheuttavat hankaluuksia ja mahdollisesti epäreilua etua toisille käyttäjille. Reaaliaikaisesta alustasta on erittäin vaikeaa luoda täysin tasapuolista kaikille.

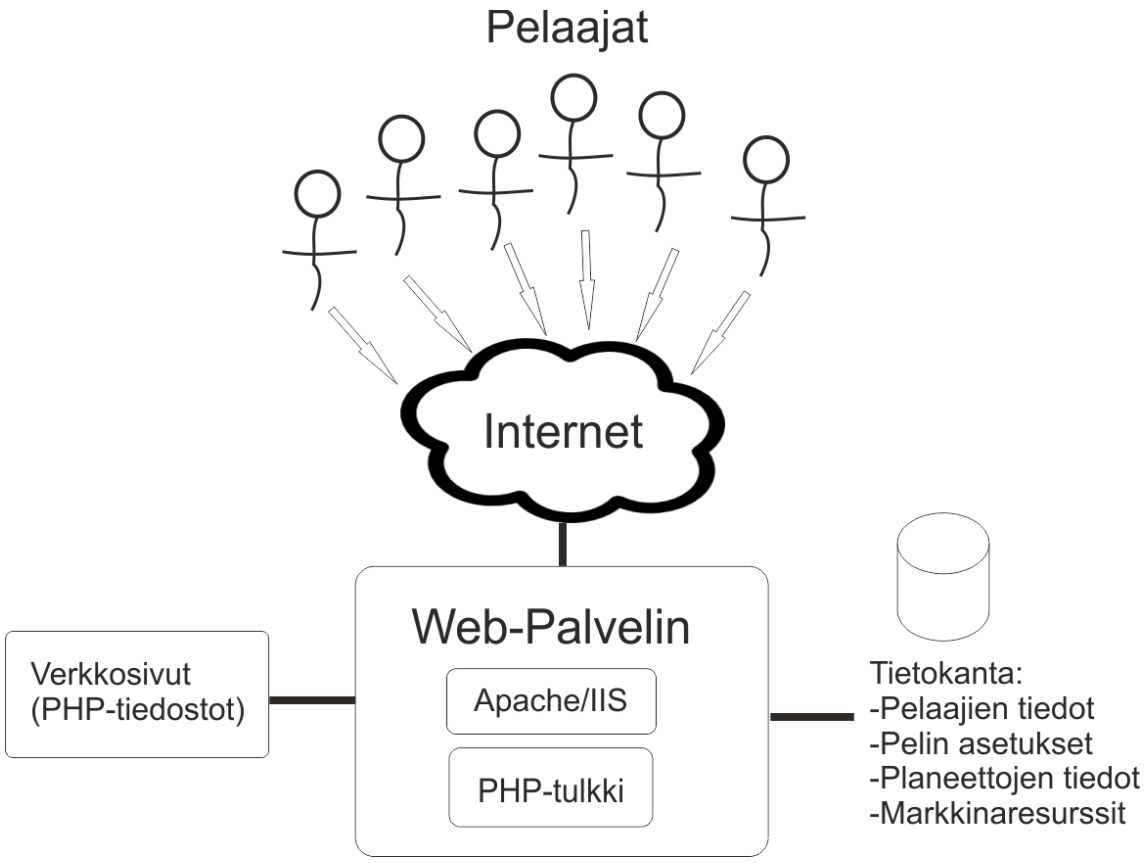
Laivastot liikkuvat planeetalta toiselle ja jokaisella planeetalla on oma osoitteensa. Mahdollisia tehtäviä laivastolle on kolmea eri tyyppiä: hyökkäys, puolustus ja siirto. Hyökätä voi kolmen ja puolustaa kuuden tapahtumalaskimen tapahtuman verran. Siirtäminen siirtää laivaston alukset kohdeplaneetan kotilaivastoon. Samaan aikaan on mahdollista hyökätä tai puolustaa useamman pelaajan ja laivaston voimin. Mahdollisia kohteita ovat kaikki planeetat, jotka eivät ole joko omassa tai oman liittouman omistuksessa. Sen lisäksi pelissä on pistemäärään perustuva hyökkäysrajoitin, joka estää hyökkäämästä liian pienien pelaajien planeetoille. Samanlainen rajoitin on rakennettu myös taistelumoottoriin siten, että planeettoja ei voi valloittaa, jos pistemäärien ero on liian suuri. Tämä siksi, että hyökkäysrajoitteet pätevät vain laivastoja lähettäessä ja laivaston liikkumisen aikana asiat voivat muuttua.

Pelaajan tavoitteena on olla pelin suurin pelaaja pistemääräisesti. Pisteet kerääntyvät pelaajan kaikista osa-alueista: aluksista, resursseista ja rakennuksista. Samanlainen tavoite on myös liitoilla, joissa pelaajien yhteispistemäärät ratkaisevat sen koon. Pelissä on myös aikarajoite kierrosten muodossa. Pelin kierros on joko ennalta määrätty aikaväli tai tapahtumalaskimen tapahtumien määrä.

Työ tehtiin pitäen mielessä, että sitä voisi käyttää myös muissa projekteissa hyväksi ja ettei se olisi pelkästään EmpireQuestia varten tehty. Taistelumoottorista piti siis tehdä erillinen osa, jolla olisi omat sääntönsä. Määrättäisiin ennalta, mitä tietoja ja missä muodossa niitä sille syötettäisiin. Palautteena se antaisi myös ennalta määrätyn muotoisen syötteen. Tämä ajattelutapa sopi myös siihen, ettei itse pelin ohjelmakoodiin ollut pääsyä vasta kuin taistelumoottorin valmistuttua.

2.2 PHP alustana

PHP on ohjelmointikielenä melko yksinkertaista. Se toimii siten, että ohjelmakoodit tallennetaan web-palvelimelle tiedostona. Kun käyttäjä pyytää sitä, PHP-tulkki ajaa kirjoitetun ohjelman (kuva 3). Sen lisäksi palvelimella on ajastettuja tehtäviä, joita ajetaan tietokannan tietojen päivittämiseen ja muun muassa taisteluiden ajamiseen. PHP:lla voi toteuttaa muitakin kuin verkkosovelluksia, mutta se on sen yleisin käyttötapa. Pelin pelaamiseen tarvitaan vain Internet-selain ja sen verkko-osoite.



Kuva 3 . Pelin sijainti ja toiminta Internetin yli

EmpireQuest pyörii web-palvelimella, jossa on PHP-tuki [5] ja MySQL-tietokanta [6]. Tietokanta sisältää kaikki pelin tiedot, kuten esimerkiksi pelaajien tilien tiedot, alusten määrät ja uutiset. Tästä seuraa muutamia etuja ja haittoja. Ensimmäisenä etuna on se, että käyttäjät tarvitsevat vain Internet-selaimen eikä muusta tarvitse välittää. Käyttäminen ei siis rajoitu käyttöjärjestelmiin tai muihin sellaisiin ohjelma-alustoihin tai -versioihin. Kaikissa nykykoneissa on vakiona Internet-selain ja useimmissa se on jopa sidottu käyttöjärjestelmän muihinkin tehtäviin, kuten tiedostojen selaamiseen. Toisena etuna on myös se, että PHP:lla on Internetissä laaja tukiverkosto ja melkein loputon määrä oppaita [5].

Haittapuolista suurin on se, että palvelinalustan yhteensopivuuden varmistaminen vaatii hieman tuntemusta. Ohjelmoijan näkökulmasta on tärkeää tietää, millainen alusta ja mitkä ohjelmistoversiot palvelimella on. Yksi yleinen ongelmatilanne on se, että ohjelmoija on tehnyt omalla testialustallaan toimivan ohjelman mutta se ei toimikaan varsinaisella tuotanto- tai julkaisupalvelimella. Syynä voi olla yksinkertainen asia, kuten väärä asetus tai sitten palvelimelta puuttuva ominaisuus tai liitännäinen, joka on lisätty

vasta uudemmassa PHP:n versiossa. Helppona ratkaisuna ei toimi sekään, että palveluntarjoaja päivittäisi ohjelmaversion, koska samalla palvelimella vanhaa ohjelmaversiota käyttävät sivut voivat lopettaa toimintansa. Tämä riippuu toki siitä, onko palvelin jaettu kuten yleensä, vai pelille omistettu, vaikka vain virtuaalipalvelimena. PHP:hen on myös saatavilla erikseen asennettavia kirjastoja, jotka sisältävät hyödyllisiä ominaisuuksia kuten esimerkiksi syötteen tarkistukseen, johon on olemassa nykyään ylläpidetty kirjasto.

2.3 PHP:n suoritusrajoitteet

Yksi PHP:n erityishuomiota ansaitseva piirre on suoritusrajoitteet. Parhaassa tapauksessa ohjelmoijalla on oma palvelin, jossa ajetaan vain omia ohjelmia. Tämä on kallista, ellei kyse ole liiketoiminnasta, jolla palvelimen ylläpitoa voi maksaa. Siksi monet päätyvät käyttämään jaettuja palvelimia, jolloin samaa palvelinta käyttää useampi käyttäjä. Käytännössä tämä tarkoittaa sitä, että käyttäjien käyttämää suoritusaikaa ja muistin käyttöastetta rajataan ja tarkkaillaan tarkoin. PHP:n vakioasetuksena on, että yhtä ohjelmaa voi pyydettäessä enintään kerrallaan ajaa 30 sekunnin ajan. Tätä aikaa voi vaihdella muokkaamalla PHP:n ominaisuuksia, olettaen että siihen on käyttäjällä oikeudet. Tämä kuitenkin tarkoittaa yleensä sitä, että palvelin on oma ja siinä ajetaan vain omia testattuja ohjelmia, sillä palveluntarjoajat harvemmin antavat lupaa peruskäyttäjille muokata ohjelmien ominaisuuksia. Palveluntarjoajat karsastavat huomattavasti suoritusaikaa vaativia ohjelmia, koska ne saattavat huonontaa muiden käyttäjien palvelun tasoa. Toisaalta se on tarpeellista, sillä jokin yhden käyttäjän huonosti tehty ohjelma voisi viedä kaikki resurssit palvelimelta tai pahimmassa tapauksessa jopa kaataa sen.

Taistelumoottori on toteutettu omaksi lohkoksi PHP-funktiona. Funktio on ohjelmapätkä, jolle voi antaa syötteenä parametreja ja se voi palauttaa arvon, olion, taulukon tai niiden muistiosoitteen viitteen tuloksena. Tällä tavoin taistelumoottoria on helppo käyttää osana kokonaisuutta, koska sitä voi kutsua sen nimellä siellä, missä sitä tarvitaan. Tämän takia on huomattava, että funktion kuten taistelumoottorin oma suoritusaika on yleensä vain osa suurempaa kokonaisuutta tai ohjelmaa. Suoritusaikaan pitää lisätä esimerkiksi tietojen hausta johtuvat viiveet, jotka voivat ruuhkautuneessa tietokannassa venähtää yllättävänkin suuriksi. Ohjelmien jatkuva ajaminen ei ole siis mahdollista ja

niitä ei voi jättää esimerkiksi odottamaan taustalle käyttäjältä uutta syötettä. Yksittäistä ohjelmaa on mahdollista ajaa täyden ajan edestä, mutta suoritusrajoitteen ylittäminen lopettaa ohjelman suorituksen. Mahdolliset tiedot ja tulokset katoavat, mikä taistelumoottorin tapauksessa olisi erittäin huono asia, koska taistelu jäisi ratkaisematta.

Muistin rajallisuuteen liittyviin ongelmiin ei työssä törmätty muuten kuin suoritusnopeuden hetkellisenä hidastumisena. Nykyiset tehokkaat palvelimet tekevät mahdolliseksi erittäin isojenkin ohjelmien ajamisen ongelmitta ja kriittisiin ongelmiin ja rajaarvoihin törmännee vasta todella massiivisten ohjelmien kanssa. PHP-ohjelmien rajattu suoritus tarkoittaa myös sitä, että ajamisen jälkeen sen käyttämät resurssit vapautetaan, mikä vähentää tietenkin muistivuotoa ja siitä aiheutuvia ongelmia palvelimella. Vanhat käyttämättömät ohjelmat eivät jää muistiin häiritsemään muita aktiivisia ohjelmia. Se, että muistivuotoa on hankala huomata tai saada aikaiseksi, ei kuitenkaan ole tekosyy tehdä ohjelmakoodista huonoa luottaen siihen, että palvelin siivoaa jäljet.

Nämä edellä mainitut seikat rajaavat hyvin paljon sen, millaisia ohjelmia PHP:llä voi tehdä. Alkuperäisen EmpireQuestin yksi suurin ongelma oli pelaaja- ja alusmäärien kasvusta johtuva rasitus palvelimelle. Taistelut oli toteutettu niin monimutkaisesti ja ajattelematta resurssien riittämistä, että palvelin kaatui sen aiheuttaman taakan alle. Palvelin yritti laskea kaikkia taisteluja samaan aikaan, samalla kun pelaajat päivittivät pelisivua aiheuttaen lisätaakkaa. Nämä vanhan pelin ongelmat sekä PHP:n suoritusrajoitteet tietäen piti taistelumoottori suunnitella niin, että pelaajien, taisteluiden tai alusten määrät eivät muodostuisi ongelmaksi. Varorajoja ei saisi edes lähestyä.

3 Mekaniikka

Kun lähtökohdat ja alustan rajoitteet tiedetään, voidaan lähteä tarkemmin käsittelemään sitä, miten taistelumoottori oikeasti ratkaisee tapahtumat. Tässä luvussa kerrotaan, miten aluksien ominaisuudet määräytyvät ja mitä ne tarkoittavat, miten alukset päättävät kohteensa sekä miten ampumisen osuminen ja järjestys lasketaan [2]. Liitteestä 1 löytyy alkuperäisen EmpireQuestin taistelun säännöt, joita lähdettiin todentamaan.

3.1 Alusten ominaisuudet

Nimi	Luokka	Tyyppi	Kohde	Panssari	Aseet	Aseen voima	Tarkkuus	Ketteryys	Toimintavuoro
Fighter	Taistelija	Normaali	Taistelija	10	3	10	50 %	50 %	2
Assault Fighter	Taistelija	Normaali	Korvetti	40	3	50	35 %	40 %	2
Chameleon	Korvetti	Normaali	Fregatti	100	4	90	35 %	30 %	2
Assault Frigate	Fregatti	Normaali	Taistelija	800	65	10	55 %	35 %	2
Guardian	Fregatti	Normaali	Risteilijä	1200	25	60	35 %	30 %	2
Assault Cruiser	Ristelijä	Normaali	Korvetti	400	10	50	35 %	30 %	2
Scorpion Cruiser	Ristelijä	Normaali	Taistelualus	800	8	200	35 %	20 %	2
Dreadnaught	Taistelualus	Normaali	Fregatti	1600	10	160	40 %	20 %	2
Mothership	Taistelualus	Normaali	Risteilijä	2500	100	30	30 %	20 %	2
Resource Freighter	Korvetti	Resurssien varastaja	(ei mikään)	250	2	0	60 %	30 %	8
Salvager	Korvetti	Kierrättäjä	(kaikki)	30	1	0	30 %	10 %	8
Stinger	Korvetti	Jäädyttäjä	Fregatti	150	3	0	30 %	40 %	6
Troop Transport	Korvetti	Valtaaja	(ei mikään)	80	2	0	30 %	25 %	7
Troop Carrier	Fregatti	Valtaara	(ei mikään)	500	4	0	35 %	25 %	7
Annexer	Ristelijä	Tuotannon varastaja	(ei mikään)	350	2	0	35 %	40 %	8
Pirate	Ristelijä	Puolustuksen tuhoaja	(kaikki)	1000	20	50	50 %	30 %	1
Ship Boarder	Ristelijä	Varastaja	(kaikki)	350	2	0	50 %	40 %	7
Orbital Ion Platform	Korvetti	Jäädyttäjä	Korvetti	400	4	0	35 %	10 %	1
Orbital Mine	Korvetti	Miina	Taistelualus	200	1	0	50 %	10 %	1
Orbital Laser Platform	Ristelijä	Normaali	Taistelija	600	70	10	40 %	10 %	1
Orbital Rocket Platform	Taistelualus	Normaali	Risteilijä	1400	10	100	50 %	10 %	1
Shield Generator	Fregatti	Ei ammu	(ei mikään)	400	1	0	0 %	10 %	N/A
Tachyon Broadcast Center	Fregatti	Ei ammu	Taistelija	200	0	0	0 %	10 %	N/A

Kuva 4 . Esimerkki alusten ominaisuuksista listattuna. Selitteet ovat taulukossa 1.

Pelissä kerätään aluksia laivastoihin ja laivastoissa voi olla useita erilaisia alustyyppejä. Aluksia kuvaamaan on luotu sarja ominaisuuksia (kuva 4). Sen lisäksi aluksilla saattaa olla erilaisia toimintatapoja. Ne voivat joko taistella tai suorittaa muita taustatehtäviä, kuten tuhottujen alusten kierrätystä. Itse ominaisuuksien arvojen sijainti riippuu itse pelistä ja sen palvelinalustasta, mutta taistelumoottori hakee ja kerää ne taulukkoon alustusvaiheessa. Taulukosta taistelumoottori sitten hakee tarvittavat arvot eri suorituksen vaiheissa.

Taulukko 1. Ominaisuuksien selitteet

Ominaisuus	**Selitys**
Nimi	Aluksen kutsuma- tai tyyppinimi.
Luokka	Kertoo aluksen malliluokan. Mahdollisia luokkia ovat taistelija, korvetti, hävittäjä, fregatti risteilijä ja taistelualus (fighter, corvette, frigate destroyer, cruiser ja battleship).
Tyyppi	Kertoo tavan, jolla alus käyttäytyy (taulukko 2).
Kohde	Kertoo aluksen ensisijaisen kohteen.
Kestävyys	Kertoo, paljonko alus kestää vahinkoa.
Aseet	Kertoo monta laukausta alus ampuu vuorossa
Aseen vahinko	Kertoo, paljonko yksi laukaus tekee vahinkoa osuessaan.
Tarkkuus	Kertoo aluksen tarkkuuden prosentteina.
Ketteryys	Kertoo aluksen ketteryyden prosentteina.
Toimintavuoro	Kertoo, millä toimintaluvulla alus toimii.

3.2 Ampuminen

Ampuminen toteutetaan hakemalla toimintavuorossa olevien alusten määrä ja kertomalla määrä aluksen ominaisuuksien mukaisella aseiden määrällä, josta saatu tulos on laukausvaranto. Laukausvaranto kontrolloi taistelumoottorin rakenteessa myös toistoa. Alukset ampuvat, kunnes vihollisia ei ole tai ammusvaranto on tyhjä. Kun laukausvaranto on laskettu, seuraavaksi haetaan aluksen tarkkuus sekä sille kohdeluokka ja kohdealuksien määrät ja ominaisuudet.

3.2.1 Eri ampumistyypit

Alusten ominaisuuksiin kuuluu, että ne voivat ampua monella eri tavalla (taulukko 2). Tämä luo peliin monimuotoisuutta ja taktikointia. Ohjelmoijan näkökulmasta eri ampumistyypit kumminkin monimutkaistavat asioita. Erilaiset ampumatyypit luovat erilaisia tiloja aluksille. Taistelumoottoriin sisälletyt tilat ovat elossa, toimivat, tuhotut ja jäädytetyt. Elossa oleva alus on vahingoittumaton, toimiva alus ei ole vielä toiminut, tuhottu alus ei enää kuulu kahteen edellä mainittuun ryhmään ja jäädytetty alus ei kykene enää toimimaan mutta on elossa. Toimivista aluksista pidetään kirjaa, jotta tiedetään ne alukset, jotka voivat vielä toimia vuorollaan. Jäätyneiden alusten tilaa tarvitsee pitää yllä siksi, että vain niitä voidaan varastaa. Vaikka jotkin alukset eivät sinänsä ammu, käsitellään niiden toiminta ampumisena toimintajärjestyksen ja onnistumislaskennan takia.

Taulukko 2. Mahdolliset eri ampumistyypit

Tyyppi	**Selitys**
Normaali	Ampuu normaalisti tuhoten kohteet.
Jäädyttäjä	Ampuu muuten normaalisti, mutta ammuksen osuessa, se jäädyttää kohteena olleen aluksen. Toissijaisena kohteena kohdistaa kaikkia jäljellä olevia aluksia.
Varastaja	Varastaa aiemmin jäädytettyjä aluksia ja siirtää ne omistajan laivastoon.
Valtaaja	Valtaa planeettoja tai vaihtoehtoisesti tuhoaa kohteen tuotantoesineitä.
Puolustuksen tuhoaja	Kohdistaa vain kohteen puolustusrakennuksia.
Kierrättäjä	Kierrättää taistelukentällä tuhottujen alusten materiaalia resursseiksi omistajille.
Tuotannon varastaja	Varastaa kohdeplaneetan tuotantoesineitä.
Resurssien varastaja	Varastaa kohteena olevan pelaajan resursseja.
Miina	Ampumisen sijaan törmää kohteeseensa tuhoten sen ja itsensä. Kohdistaa vain ensisijaista kohdetta.

3.2.2 Ampumisen kohdistaminen ja osumistarkkuus

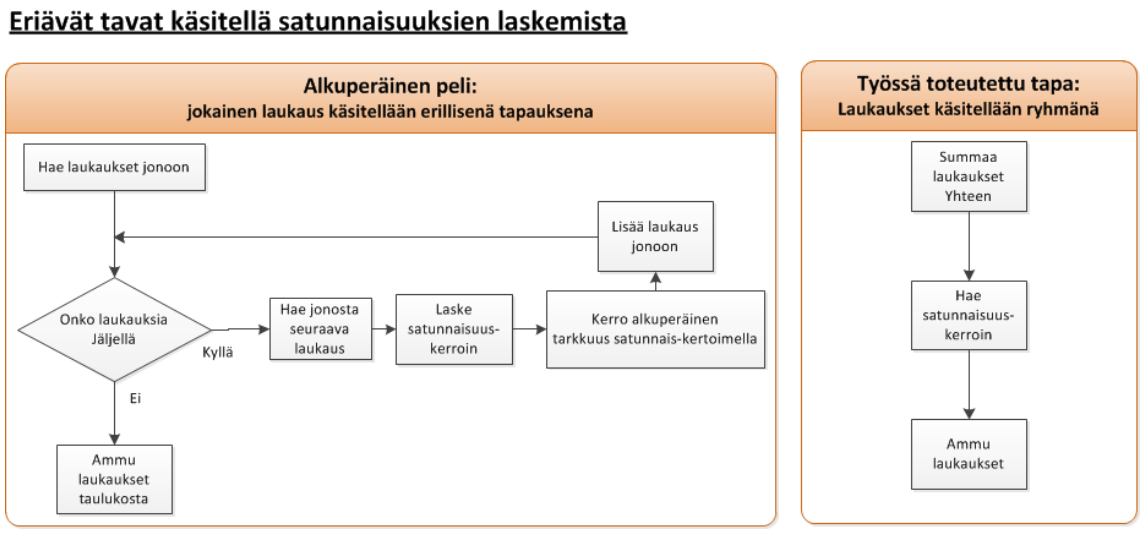
Vihollisaluksia ampuvalla aluksella on ensisijainen kohdeluokka. Jos kohdeluokkaan kuuluvia aluksia ei ole, arvotaan seuraava kohdeluokka jäljellä olevien vastapuolen aluksien pohjalta. Tämä toistuu, kunnes joko ammusvaranto on tyhjä tai vihollisalukset on tuhottu. Toissijaista kohdetta ammuttaessa tarkkuus vähenee ennalta määritetyn määrän verran, joka vakiona on 50 prosenttia. Tällä tavoin luodaan tarvetta kohdistamiselle, sillä jos tarkkuus ei vähenisi, riittäisi yksi alustyyppi tuhoamaan useimmat vastustajan alukset puhtaan määrän ja ampumisjärjestyksen perusteella.

Aluksien laukausten osumistarkkuutta varten tarvitaan jokin määre. Sitä varten peliin on tehty matemaattiset kaavat, joiden avulla laukauksille saadaan laskettua todennäköisyys, jolla se osuu kohteena olevaan alukseen. Jokaiselle eri ampumistyypille on luotu kaava, jossa huomioidaan tarvittaessa kohteen ketteryys (taulukko 3).

Taulukko 3. Eri ampumistyylien kaavat

Tyyppi	**Kaava**
Normaali, jäädyttäjä tai varastaja	15 %+(tarkkuus-(kohteen ketteryys/2)) %
Valtaaja	10 %+(tarkkuus) %
Puolustuksen tuhoaja	15 %+(tarkkuus-10%) %
Tuotannon varastaja	10 %+(tarkkuus-17.5%) %
Resurssien varastaja	10 %+(tarkkuus-17.5%) %

Alkuperäisen pelin ominaisuuksiin kuului, että ampumisessa oli satunnaisuutta ja se oli toteutettu laskemalla satunnaisuus jokaiselle laukaukselle erikseen (kuva 5). Työssä päädyttiin yksinkertaistamaan toimintaa laskemalla satunnaisuus kerran ampuvan ryhmän tarkkuuteen ja kohteina olevien aluksien ketteryyteen. Satunnaisuuden suuruus on säädettävissä, mutta normaalisti heilahteluväli on 5 prosenttia jompaankumpaan suuntaan. Satunnaisuuden tarkoitus ei ole ratkaista taisteluita, vaan tuoda vaihtelua taisteluihin. Tällä tavoin taistelujen lopputulos ei ole täysin ennakkoon simuloitavissa ja mielenkiinto tulosta kohti säilyy.



Kuva 5 . Satunnaisuuksien laskeminen.

Se miksi päädyttiin laskemaan satunnaisuus koko ryhmälle kerralleen sen sijaan, että laskettaisiin jokaiselle laukaukselle oma tarkkuutensa (kuva 5), voidaan selittää esimerkin avulla. Otetaan esimerkiksi tikanheitto tikkatauluun. Heittäjä osuu keskimääräisellä tarkkuudella viitoseen. Hänen tuloksensa voi vaihdella tasaisesti 2 pisteen verran molempiin suuntiin, jolloin tulosväliksi saadaan 3-7. Jos hän heittää vain kerran, hän voi saada satunnaisen tuloksen, joka on 3-7 väliltä. Jos hän heittää useamman kerran, saadaan tulosten keskiarvoksi 5. Hänen tuloksensa siis tasoittuu useamman kerran heitettäessä. Tätä voi verrata myös ampumiseen. Jos jokaiseen laukaukseen laskettaisiin satunnainen muutos ennalta määritellyltä väliltä, tasoittuisi yhdistetty osumatarkkuus lopulta alkuperäiseen tarkkuuteen. Tulokset olisivat suurin piirtein samat, vaikka yksittäinen laukaus osuisi tarkemmin ja jokin toinen epätarkemmin.

Asiaa voi myös tarkistella laskennan tehokkuuden näkökulmasta. Otetaan havainnollistavaksi esimerkiksi 1000 hävittäjän ampuminen. Olkoot yhdellä hävittäjällä 2 laukausta ja osumistarkkuus on 0,50 eli 50 prosenttia. Viiden prosentin satunnaisuus lasketaan kertomalla alkuperäinen tarkkuus arvotulla luvulla 0,95 ja 1,05 väliltä. Laukauksia on 2000. Tarkkuuden heilahteluväliksi saadaan siis 0,475-0,525 eli 47,5-52,5%. Tämän jälkeen voidaan ampuminen käsitellä kahdella tapaa: joko koko hävittäjän ryhmälle arvotaan kerran tarkkuus, jota kaikki laukaukset käyttävät tai jokaiselle 2000 laukaukselle lasketaan erikseen tarkkuus. Jos osumistarkkuus lasketaan jokaiselle laukaukselle erikseen, voidaan huomata, että kaikkien arvottujen tarkkuuksien keskiarvo on lähellä 50 prosenttia, eli alkuperäistä tarkkuutta. Tarkkuuksien erot siis tasoittuvat. Jos taas

tarkkuus arvotaan vain kerran koko ryhmälle, saadaan 1 satunnainen luku, joka voi erota alkuperäisestä tarkkuudesta. Huomataan siis, että jokaiselle laukaukselle erikseen laskettaessa hyötyä ei saavuteta, mutta suoritetaan 4000 turhaa laskutoimitusta, jotka koostuvat 2000 satunnaisuuden laskemisesta ja 2000 kertolaskusta. Ryhmälle laskettaessa tehdään 2 laskutoimitusta ja satunnaisuus toteutuu nähtävästi. Tämän esimerkin 1000 hävittäjää on pieni alusmäärä pelissä, laukausten määrät nousevat huomattavan nopeasti.

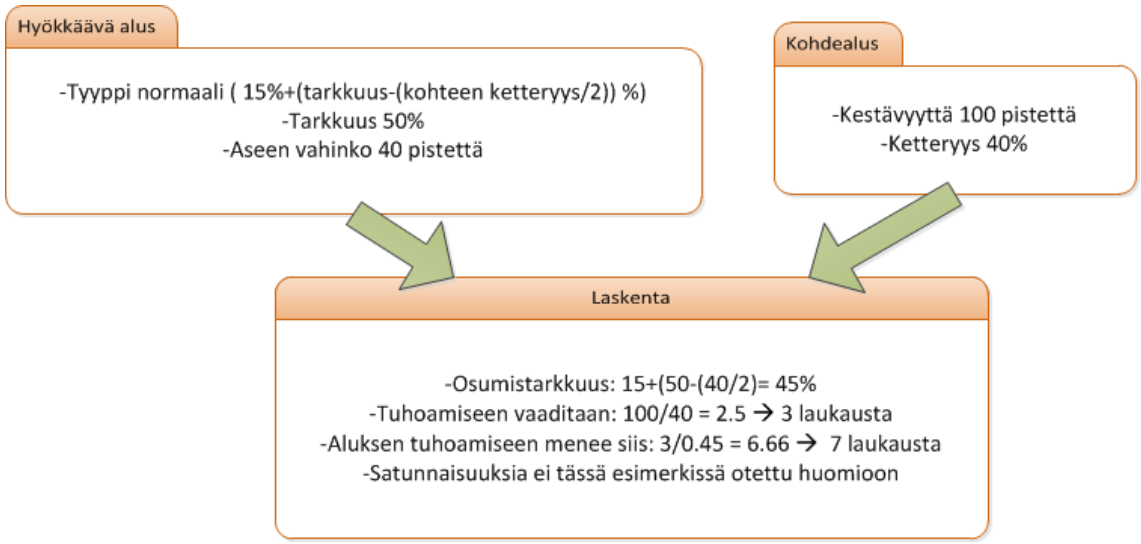
Siis jokaiselle laukaukselle eri todennäköisyyden laskeminen on turhaa, koska tulos tasautuu alkuperäisen tarkkuuden tienoille. Olisi siis aivan sama, jos satunnaisuutta ei olisi ollenkaan. Päätymällä ratkaisemaan satunnaisuus kerrallaan koko alustyypille satunnaisuus toteutuu hieman paremmin, vaikutus näkyy paremmin. Lisäetuna tästä toteutustavasta saadaan vielä eräänlainen immuniteetti alusten määrän kasvun aiheuttamille ongelmille. Jos aluksia ja laukauksia kohdeltaisiin yksittäisinä tapauksina edellä mainitulla tavalla, nousisi palvelimen suorittamien laskutoimitusten määrä eksponentiaalisesti, koska aluksia kohdellaan ryhminä, on laukausten määrä vain yksi muuttuja, ampuminen yksinkertainen laskutoimitus. Alusten määrä vaikuttaa vain muuttujan arvoon eikä lisää laskutoimitusten määrää. Alkuperäisen pelin kaatumisen alusten määrien kasvaessa osasyynä olikin juuri syynä se, että laskutoimitusten määrä nousi eksponentiaalisesti alusten määrän kasvaessa, koska aluksilla oli useita aseita ja jokaiselle piti laskea satunnainen tarkkuus.

3.2.3 Ampumisen vahinkojen laskeminen

Kun kohdeluokat ja niiden alusmäärät ovat selvillä, on aika selvittää, miten paljon tuhoa kukin alus saa aikaiseksi. Kukin alusryhmä, eli ryhmä samoja aluksia, ampuu sen ajan, kun laukausvarannossa on laukauksia jäljellä. Tämän jälkeen mahdollisia ratkaisuja voisi olla monta. Ennen sitä on kumminkin tarpeellista selittää, miten aluksen tuhoamiseen vaadittavien laukauksien lukumäärä lasketaan.

Laukauksia kohdellaan vain kokonaisina. Otetaan esimerkiksi tilanne, jossa tarkoituksena on tuhota yksi kohdealus, mutta selkeyden takia hyökkääjän tarkkuuden ja puolustajan ketteryyden satunnaisuutta ei oteta huomioon. Hyökkäävän aluksen tyyppi on normaali, sen tarkkuus on 50 % ja aseen vahinko 40 pistettä. Puolustavan aluksen

kestävyys on 100 pistettä ja sen ketteryys on 40 %. Hyökkäävän aluksen tarkkuudeksi saadaan kaavalla 45 % ja 100 pisteen tuhoamiseen tarvitaan 3 osunutta laukausta (kuva 6). Laukauksia kohdellaan kokonaisina, joten 3 laukauksen optimaalisesta 120 pisteen vahingosta jää 20 pistettä yli, joka jää käyttämättä. Tällä tavoin luodaan aseisiin erilaisuutta ja eri käyttötarkoituksia. Hyttysen voi tappaa kranaatilla, mutta se on tarpeettoman suurta voimankäyttöä. Puolustavaan alukseen täytyy myös osua, joten osumistarkkuuden avulla saadaan ampumattomien laukausten määrä, jotka tarvitaan yhden aluksen tuhoamiseen, joka tässä tapauksessa on seitsemän (kuva 6).

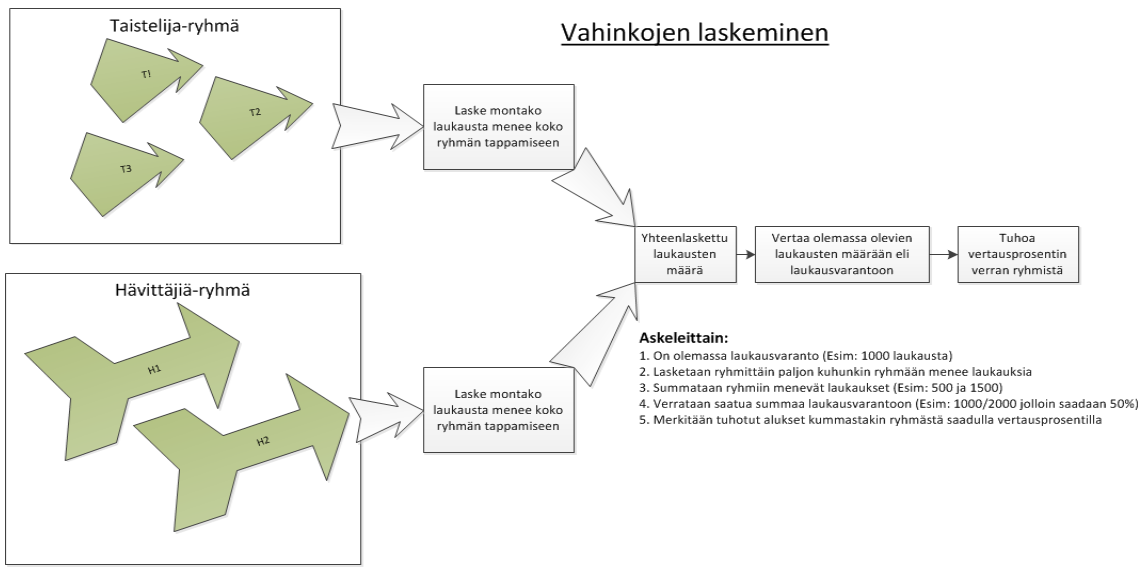


Kuva 6 . Yhden aluksen tuhoutumisen laskeminen ilman satunnaisuutta

Teoriassa hienoin tapa ja samalla ainut todellisuuden mukainen tapa toteuttaa samanaikainen ampuminen olisi, että samalla toimintavuorolla toimivien sekä hyökkäävän että puolustavan puolen alusten laukaukset järjestäisi jonoon ja ne ammuttaisiin vuorotellen samaan aikaan tuhoten kohdealukset, joiden kestävyys loppuisi. Tuhotut alukset eivät enää voisi ampua, ne poistettaisiin jonosta tarpeen mukaan. Jopa kohdealuksista voisi luoda oman kirjapidon, jossa jokaisen kestävyyden tasoa pidettäisiin silmällä. Tapa on kuitenkin liian monimutkainen jo senkin takia, että laskutoimitusten määrä nousisi erittäin nopeasti jo pelkkien satunnaisten alkutoimien laskemisen takia. Mahdollisia variaatioita ideasta on monta, mutta riittäköön hienoin tapa esimerkkinä teoriasta. Seuraavaksi kerrotaan tapa, joka taistelumoottorissa toteutettiin.

Taistelumoottorin ratkaisu perustuu hyvin luvussa 3.2.2 mainittuun tikanheittoesimerkkiin, jossa tarkkuus tasoittuu, mitä enemmän toistoja on. Sen pohjalta lähdettiin kehittämään tapaa, jolla turhien laskujen määrän saa mahdollisimman vähäiseksi. Ensimmäisenä keinona ryhmitetään taistelua eikä lähdetä erittelemään aluksia. Ampuvalla alusryhmällä on siis yhteinen laukausvaranto.

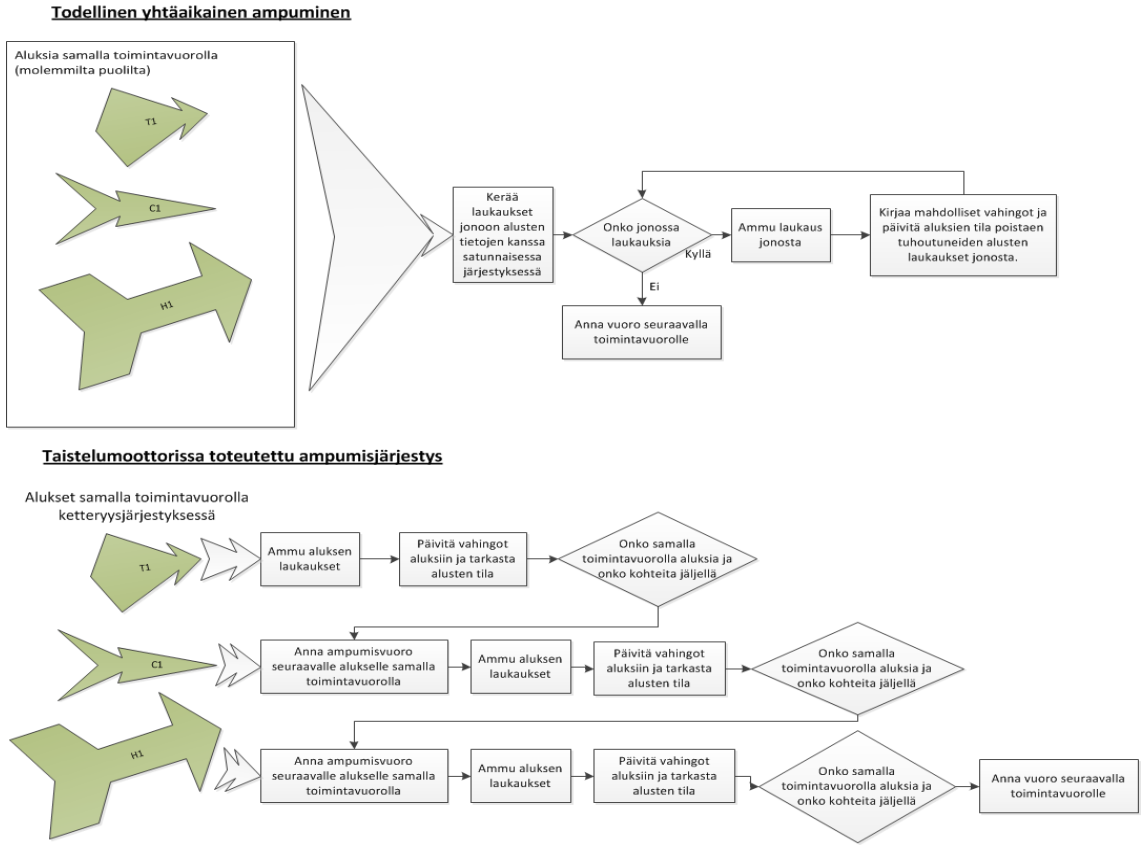
Seuraava ongelma on, että jos kohteena on useaa alustyyppiä, joilla on eriävät kestävyydet sekä ketteryydet, miten päätetään, kuinka laukaukset jakautuvat eri alustyyppien kesken. Ongelma ratkaistiin kääntämällä ajattelu lopusta alkuun eli laskemalla kaikkien kohteena olevien aluksien tuhoamiseen tarvittavan laukausten määrä. Saatua lukemaa pystyy sitten vertaamaan saatavilla olevaan laukausvarantoon ja päättelemään, tuhoutuvatko kaikki kohdealukset vai vain osa niistä (kuva 6). Jos laukausvaranto on suurempi, voi tarvittavien laukauksien määrän vähentää suoraan ja jatkaa ampumaan seuraavaa satunnaista kohdeluokkaa luvussa 3.2.2 selitetyllä tavalla. Jos laukausvaranto on yhtä suuri kuin tarvittava laukausten määrä, voi kohteet merkitä tuhotuksi, laukausvarannon käytetyksi ja antaa ampumisvuoron seuraavalle. Viimeisenä vaihtoehtona on, että laukausvaranto on pienempi kuin kaikkien alusten tuhoamiseen tarvittava määrä, jolloin aluksista merkitään tuhotuksi laukausvarannon ja tarvittavien laukausten suhdeluvun osoittama määrä, ampumisvuoro annetaan seuraavalle alusryhmälle. Samalla myös varmistetaan se tapaus, jos kohteena on useita alusryhmiä, että niistä tuhoutuu aluksia oikeassa suhteessa alkuperäiseen määrään ja muihin alusryhmiin.



Kuva 7 . Vahinkojen laskeminen

3.2.4 Samanaikainen ampuminen

Samanaikaisen ampumisen käsitellään sääntöjen pohjalta. Asian voisi puhtaasti hoitaa alusten toimintavuorojen määräämässä järjestyksessä ja varmistamalla ettei kahdella aluksella ole samaa toimintavuoroa. Jos kumminkin on tilanne, että kahdella eri alustyypillä on sama järjestys, voisi esimerkiksi ajattelua jatkaa, että paremmalla ketteryyden omaavat alukset saavat ampua ensin. Taistelumoottori toimii juuri tällä tavoin, mutta vielä hieman yksinkertaisemmalla säännöllä: mitä aiemmin on merkittynä ominaisuustaulukkoon (tarkemmin luvussa 3.2), sitä ennemmin saa ampua. Eli alukset on merkitty taulukkoon ketteryysjärjestykseen (kuva 7). Tällä tavoin vältytään turhalta vertailulta, koska asia on päätetty kiinteästi. Tästä muodostuva haitta on se, että taulukon järjestyksen vaikutus taistelumoottorin käyttäytymiseen täytyy ymmärtää, jos vaihtaa alusten ominaisuuksia. Satunnaiselle käyttäjälle asia olisi helpompi, jos funktio suorittaisi ampumajärjestyksen priorisoinnin, koska silloin hänen ei tarvitsisi välittää asiasta. Toisaalta on jopa hyvä asia, että käyttäjä perehtyy siihen, miten taistelumoottori käyttää alusten ominaisuuksia, ennen kuin rupeaa niitä muuttamaan.



Kuva 8 . Eri tapoja toteuttaa ampumisjärjestys.

Entä mitä käytännössä tarkoittaa tai mitä etua on siitä, että saa ampua ensin jos ollaan samalla toimintaluvulla? Tällä hetkellä siitä saavutetaan se hyöty, että saa ampua omaa ensisijaista kohdettaan ensin, ennen kuin muut alukset ehtivät tuhota ne. Mahdolliset vahingot vastapuolelle merkitään väliaikaiseen taulukkoon talteen, jotka huomioidaan, kun etsitään seuraavalle ampujalle kohdetta. Vasta kun toimintavuoro loppuu, sijoitetaan vahingot väliaikaisesta taulukosta varsinaisiin olemassa olevien alusmäärien taulukoihin. Tässä kohdin on tärkeää muistaa päivittää kaikki tarvittavat taulukot, kuten esimerkiksi jäätyneiden ja ampuvien alusten nykyinen tilanne.

Käyttäytymistä on helppo muuntaa. Tarvitsee vain siirtää väliaikaisten taulukon päivittämistä eri kohtaan taistelumoottorin rakenteen sisässä. Esimerkiksi sen voisi siirtää toimimaan jokaisen eri alustyypin ampumisen jälkeen, jolloin ensimmäisenä ampunut saattaisi vaikuttaa jopa samalla toimintavuorolla myöhemmin ampuviin aluksiin. Tästä kerrotaan tarkemmin luvussa 4, jossa selitetään funktion rakennetta. Mutta kuten aiemmin mainittiin, kyse on hyvin pitkälti säännöistä ja niiden noudattamisesta. Kun säännöt tiedetään, osataan myös niiden pohjalta tehdä ominaisuudet aluksille, jotta alukset käyttäytyvät halutulla tavalla.

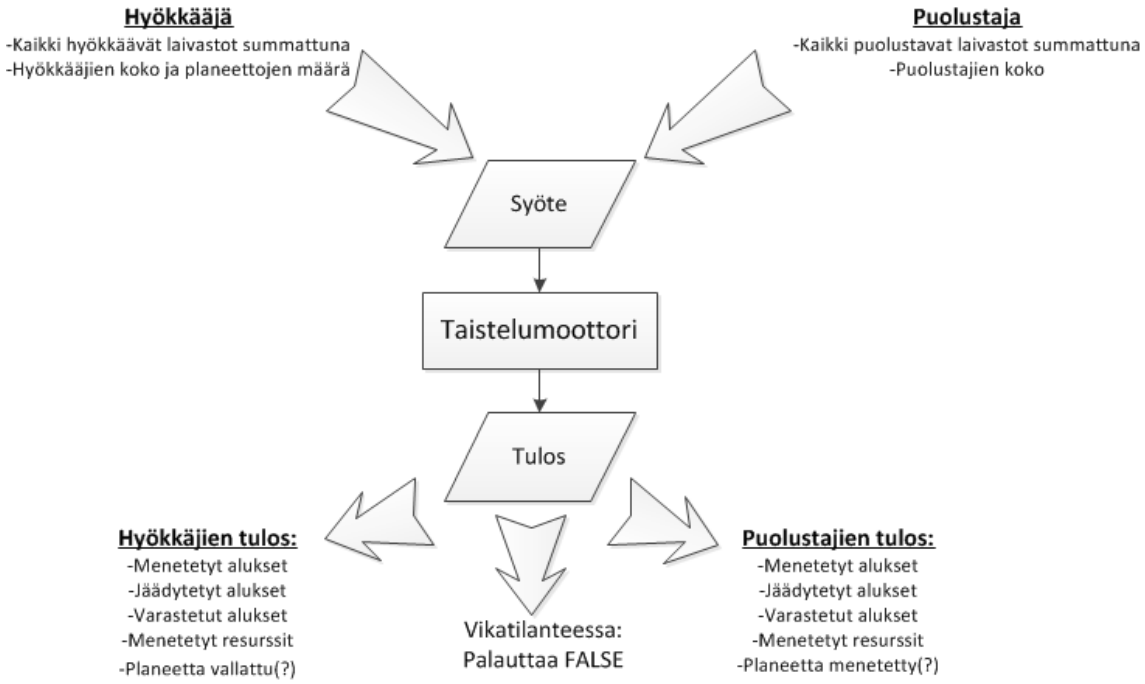
4 Taistelumoottorin rakenne

Edellisessä luvussa (luku 3) käsiteltiin, miten peli laskee ja käsittelee tarvittavat arvot. Tässä luvussa keskitytään kertomaan tarkemmin, miten taistelumoottori pyörii läpi tehtävänsä eli taistelun tuloksen ratkaisemisen.

4.1 Syöte- ja palautearvot sekä niiden tarkistus

Taistelumoottori on funktio ja sellaisena sen ominaisuuksiin kuuluu, että se voi saada parametreinaan monta muuttujaa, mutta palauttaa vain yhden arvon. Tämä arvo voi olla mikä vain yksittäinen muuttuja, olio tai vaikka taulukko. Taistelumoottori ottaa syötteenä taulukon yhtenä parametrina ja palauttaa tuloksen myös taulukkona. Syötteeseen sisällytetään kaikkien paikalla olevien alusten ja rakennuksien määrä sekä osallistujien pistemäärät ja planeetan resurssikenttien määrä (kuva 9). Molemmissa taulukoissa tietojen paikat ovat ennalta määrättyjä, mikä heikentää dynaamisuutta. Oli kui-

tenkin tarpeellista toteuttaa syöte tässä muodossa, koska peli, jolle taistelumoottoria tehtiin, sisälsi kiinteitä arvoja. Esimerkiksi rakennusten tunnistenumerot oli ohjelmoitu kiinteästi peliin niin, ettei niiden vaihtaminen onnistunut helposti. Numeron muuntaminen olisi tarkoittanut kaikkien pelin sivujen tarkastamista ja korjaamista. Liitteessä 3 on tarkemmat tiedot syöte- ja palautetaulukoiden ennalta määrätystä järjestyksestä.



Kuva 9 . Tietojen käsittely taistelumoottorissa

Kaikkiin hyviin funktioihin liittyy syötearvojen oikeamuotoisuuden tarkastus ja mahdollisten virhetilanteiden tulkinta ja niiden jälkeen oikein toimiminen. Virheellinen data voi aiheuttaa kahden kaltaisia seuraamuksia. Lievempi tapaus on, että taistelua ei tapahdu ja funktio palauttaa heti virheenkoodin kutsujalle. Hyvää tässä on se, etteivät pelaajat menetä mitään. Huonoa on se, ettei kukaan myös saavuta mitään sitä, mitä taistelussa oli tarkoituksena saavuttaa. Pahempi tapaus olisi, jos taistelun laskeminen keskeytyisi kesken laskujen ja tulos olisi korruptoitunut eli väärä. Kummassakin tapauksessa on kuitenkin yhteistä se, ettei taistelua voi ajaa uudelleen jälkikäteen, koska tilanne on saattanut jo muuttua. Pelaajat ovat saattaneet kutsua laivastoja takaisin, pisteiden määrät ovat voineet muuttua, resursseja on voitu käyttää ja niin edespäin. Jos kuitenkin oletetaan, että syöte on oikea, laaja testaus on oikea tapa varmistaa, että funktio toimii halutulla tavalla. Tästä lisää luvussa 5, jossa käsitellään taistelumoottorin testausta tarkemmin.

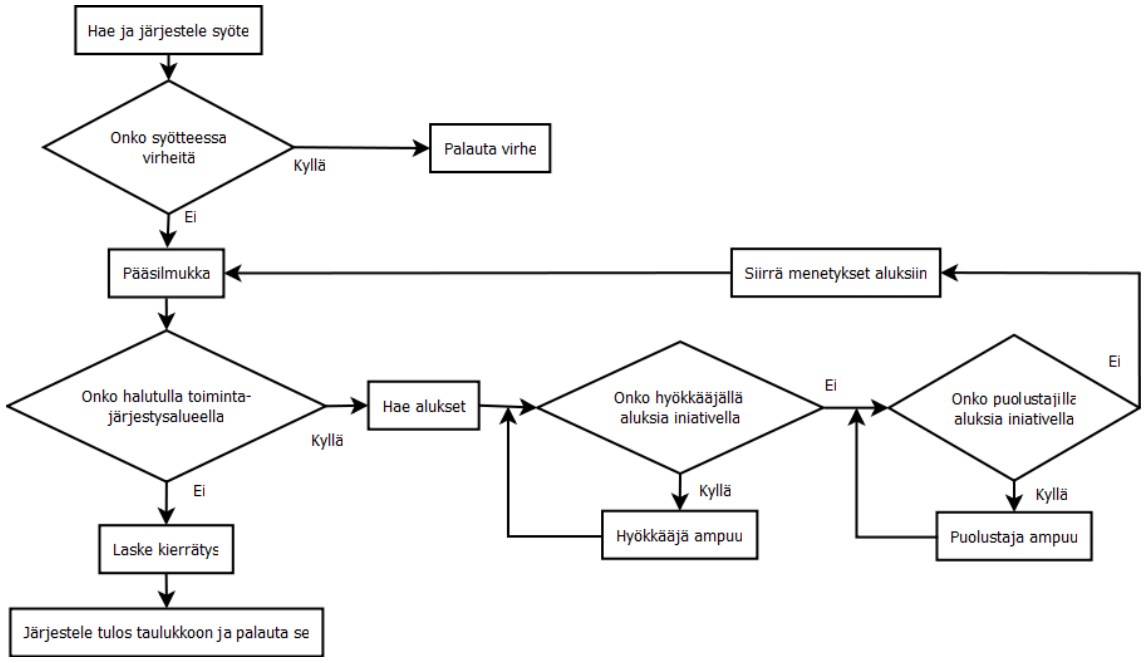
Taistelumoottori voi vain tarkistaa saadun syötteen oikeamuotoiseksi, mutta sen sisällön oikeutta se ei voi tietää. Taistelumoottorin tehtävänä ei ole tietää tai tarkkailla pelin tietokantaa. Sitä ei voitu rakentaa sillä tavoin, että se tietäisi, ovatko esimerkiksi jonkin laivaston alusmäärät tai pelaajan pisteet oikein ja siksi se tarkistaa vain, että saatu syöte sisältää vain positiivisia kokonaislukuja eikä mitään muuta. Kaiken muun tarkistaminen jää kutsuvalle ohjelmalle tai tulkille, joka taistelumoottoria käyttää.

Virhetilanteessa, jossa sille syötetään virheellistä dataa, funktio palauttaa yksinkertaisesti boolean-arvon ”false”. Kutsuvan ohjelman pitäisi siis osata huomata tämä ja tehdä oikeat jälkitoimet. Kun tiedetään pelin päivittyvän automatisoidun tapahtumalaskimen mukaan (lisää luvussa 3.1), asia onkin hankala. Virheet voisi ilmoittaa erilliselle sivulle, josta pelin järjestyksenvalvojat voivat tapauskohtaisesti päättää toimistaan tai vaikka automatisoidulla anteeksipyynnöllä pelaajille. Mitään helppoa tapaa ei ole keksitty, jolla tilanteet saisi automaattisesti analysoitua ja korvattua pelaajille. Jos tällainen tapa olisi, se olisi myös erittäin käytännöllinen testauksessa. Tilanteessa saatettaisiin päätyä olemaan hyvin lähellä tekoälyn luomista, mutta paras tapa lienee sopia yhteisön kanssa jokin hyväksytty tapa käsitellä virhetapaukset.

Yksi tärkeimmistä asioista tällaisessa matemaattisessa algoritmissa on muistaa tyhjien tai nolla-arvojen käsittely. Lisähankaluutta aiheuttaa se, ettei näitä kahta arvoa välttämättä saa huomioitua samalla kertaa, koska ne eivät ole sama asia kaikkien funktioiden kannalta. Jotkin niistä kohtelevat nolla-arvoa asetettuna arvoja ja toiset niin sanottuna null-arvona, joka tarkoittaa asettamatonta arvoa. Käsittely pitää muistaa tehdä heti, sillä aiheutuneiden ongelmien etsiminen jälkikäteen toimivasta ohjelmasta on hankalaa. Yksi yleinen nolla-arvojen aiheuttamista vioista on ikuinen silmukka. PHP osaa joissain tapauksissa varoittaa nolla-arvojen väärästä käytöstä esimerkiksi jakolaskujen yhteydessä. Toisaalta se ei puutu nolla-arvoihin esimerkiksi silmukoiden ehdoissa. Kyseessä lienee ohjelmointitekninen asia, sillä voi olla, että jotain silmukkaa halutaan ajaa asettamattoman ajan verran. Toisaalta luvussa 2.3 mainitut PHP:n suoritusrajoitteet lopettavat loputtomat silmukat suoritusajan loputtua.

4.2 Taistelumoottorin toiminta

Tässä luvussa käsitellään taistelumoottorin rakennetta (kuva 10) ja käydään läpi miten taistelumoottori käy taistelun käsittelyn läpi askel askeleelta. Samalla käydään läpi rakenteellisia ratkaisuja.



Kuva 10 . Taistelumoottorin toiminta tiivistettynä vuokaavioon

Taistelumoottorin ensimmäinen tehtävä on alustaa tarvittavat muuttujat ja asetukset sekä järjestellä saatu syöte: syötetaulukko puretaan erillisiksi hyökkääjän ja puolustajan taulukoiksi ja muuttujiksi. Syötteiden arvot tarkistetaan positiiviksi kokonaisluvuiksi ja tarvittaessa palautetaan virhe. Tarkistuksen jälkeen alustetaan uudet taulukot molempien puolien menetyksille, jäädytetyille ja varastetuille aluksille, koska näitä tarvitaan viimeistään siinä vaiheessa, kun tuloksia palautetaan takaisin kutsujalle.

```
***battleEngine.php
<?php
include_once 'haeOminaisuudet.php'; //Hakee alusten ominaisuudet

//*******Pääfunktio**********//
function battleEngine($syöte) {
/*Asetukset*/
$alinToimintavuoro=1; //alin mahdollinen toimintavuoro
$ylinToimintavuoro=21; //ylin mahdollinen toimintavuoro
$toissijainenTarkkuus=0.5; //toissijaisen ampumisen tarkkuusvähennys prosentti
$kierrättäjänKapasiteetti=10000; //kierrättäjän kapasiteetti
$kierrätysOsuus=0.4; //kierrätettävän materiaalin osuus tuhottujen alusten resurssimäärästä
$rvKapasiteetti=1000; //resurssien varastajan kapasiteetti

checkInput($input); //tarkista syöte

//järjestele syöte ajonaikaisiin muuttujiin ja taulukoihin

//pääsilmukka, käy läpi asetetun toimintavuoroalueen
for($toimintavuoro=$alinToimintavuoro; $toimintavuoro<=$ylinToimintovuoro; $toimintavuoro++){
    //tarkista puolien alusten tilanne ja hae alukset jolla toimintavuoro

        if(onko kummallakaan osapuolella ampuvia aluksia toimintavuorolla)
        {
            //aloitetaan toimintavuorolla ampuvien aluksien ampuminen ensimmäisestä taulukon aluksesta
            foreach($toimintavuoronAlukset as $alusID) {
                //aseta x ja y sen mukaan onko puolilla ampuvaa alusta.
                //Sen perusteella ajetaan switch-case jossa ammutaan molemmat, toinen tai ei kumpikaan puoli/puolet.
                for($puoli=$x; $puoli<=$y; $puoli++){   a
                    switch($puoli) { //case 0: hyökkääjä , case 1: puolustaja
                        /*********Hyökkääjän ampumistapaukset*********/
                        case 0:
                            //haetaan ampuvan aluksen tiedot (mm. $alusTyyppi) ja laske ammusvaranto

                            if($alusTyyppi=="Normaali") {
                                if(jos on vastapuolella on ensisijaista kohdeluokkaa ammuttavaksi) {
                                    //ammu kohdetta
                                }
                                while(jos vihollisia on jäljellä && ammusvaranto >0) {
                                    //etsi uusi kohdeluokka jäljellä olevista vastustajan aluksista ja ammu niitä
                                }
                            }

                            elseif($alusTyyppi=="Puolustusten tuhoaja"){
                                if(jos on puolustusrakennuksia){
                                    //ammu puolustusrakennuksia
                                }
                            }

                            elseif($alusTyyppi=="Jäädyttäjä") {
                                if(jos on vastapuolella on ensisijaista kohdeluokkaa ammuttavaksi) {
                                    //jäädytä vastapuolen aluksia
                                }
                                while(vastapuolella on jäädytettäviä aluksia && laukausvaranto >0) {
                                    //etsi uusi kohdeluokka jäljellä olevista vastustajan aluksista ja jäädytä niitä
                                }
                            }

                            elseif($alusTyyppi=="Resurssikenttien varastaja"){
                                //varasta resurssituotantoesineitä jos mahdollista
                            }
```

Kuva 11 . Ensimmäinen osa taistelumoottorin tiivistetty pseudo-ohjelmakoodista

```
62
63                          elseif($alusTyyppi=="Resurssien varastaja"){
64                              //varasta resursseja jos mahdollista
65                          }
66
67                          elseif($alusTyyppi=="Varastaja"){
68                              if(vastustajalla on jäädytettyjä aluksia) {
69                                  //varasta aluksia
70                              }
71                          }
72
73                          elseif($alusTyyppi=="Miehistönkuljetusalus"){
74                              //yritä varastaa planeetta tai tuhoa resurssikenttiä
75                          }
76                      break;
77                      /**********Puolustajan ampumistapaukset***************/
78                      case 1:
79                          //haetaan ampuvan aluksen tiedot (mm. $alusTyyppi) ja laske ammusvaranto
80
81                          if($alusTyyppi=="Normaali") {
82                              if(jos on vastapuolella on ensisijaista kohdeluokkaa ammuttavaksi) {
83                                  //ammu kohdetta
84                              }
85                              while(jos vihollisia on jäljellä && ammusvaranto >0) {
86                                  //etsi uusi kohdeluokka jäljellä olevista vastustajan aluksista ja ammu niitä
87                              }
88                          }
89
90                          elseif($alusTyyppi=="Miina"){
91                              //Ampuu vain ensisijaista kohdetta ja tuhoaa kohteensa osuessaan samalla tuhoutuen itse
92                          }
93
94                          elseif($alusTyyppi=="Jäädyttäjä") {
95                              if(jos on vastapuolella on ensisijaista kohdeluokkaa ammuttavaksi) {
96                                  //jäädytä vastapuolen aluksia
97                              }
98                              while(vastapuolella on jäädytettäviä aluksia && laukausvaranto >0) {
99                                  //etsi uusi kohdeluokka jäljellä olevista vastustajan aluksista ja jäädytä niitä
100                             }
101                         }
102
103                         elseif($alusTyyppi=="Varastaja"){
104                             if(vastustajalla on jäädytettyjä aluksia) {
105                                 //varasta aluksia
106                             }
107                         }
108                     break;
109                 }
110             }
111             //Aseta väliaikaiset tulostaulukot oikeisiin alusten tilataulukkoihin.
112         }
113     }
114 }
115
116 if(jos jommalla kummalla puolella on kierrättäjiä) {
117     //laske kierrätettyjen resurssien määrä
118 }
119
120 return $tulos; //Järjestä tulos ajon aikaisista muuttujista ja palauta se.
121 }
122 ?>
```

Kuva 12 . Toinen osa taistelumoottorin tiivistetty pseudo-ohjelmakoodista

Taistelua pyörittää juuritasolla yksinkertainen for-silmukka, jonka iteraatiot tai toistoluvut ovat yhtä kuin alusten toimintajärjestysluvut (kuva 11 ja 12). Silmukka aloittaa toimintaluvusta 1 ja päättyy haluttuun lukuun. Rakenteen yksi etu on se, että toimintalukujen lisääminen on helppoa. Kun toimintaluku on tiedossa, voidaan alusten ominaisuuksista hakea alukset, jotka tällä toimintaluvulla ampuvat. Näiden perusteella taas haetaan tiedot hyökkääjän ja puolustajan kyseisten alusten määristä. Saaduista uusista taulukoista voi sitten päätellä, onko aluksia molemmilla puolilla, vain toisella puolella tai ei ollenkaan. Tästä saadaan tapauskohtainen ratkaisu, jossa taistelumoottori käy läpi vain olemassa olevat alukset. Etuna saavutetaan parempi virhekestävyys ja myös nopeampi suoritus.

Seuraavaksi vuorossa on ampuminen. Hyökkääjä ja puolustaja ampuvat tapauksien perusteella tai eivät ollenkaan. Tästä voikin jo päätellä, että niillä on erilliset lohkot ohjelmassa. Tämä siksi, koska eri ampumistyypit saattavat toimia eri tavoin hyökkääjän ja puolustajan laivastossa. Esimerkiksi puolustuksen tuhoajat puolustajan laivastossa eivät tee mitään ja hyökkääjän laivastossa ei voi olla miinoja, koska ne ovat puhtaasti puolustusrakennelmia.

Hyökkääjä ampuu aluksensa ensin, tulokset laitetaan väliaikaiseen taulukkoon. Tämän jälkeen puolustajan vastaava alus ampuu, sen tulokset tallennetaan eri väliaikaiseen taulukkoon. Jos samalla toimintaluvulla on useampia aluksia, luovutetaan vuoro niille. Kun toimintaluvulla ei ole enää ampuvia aluksia, siirretään väliaikaisten taulukoiden mukaiset tulokset oikeisiin taulukoihin ja väliaikaiset taulukot nollataan. Vuoro annetaan seuraavalle toimintavuorolle, ellei nykyinen toimintavuoro ollut viimeinen halutulla toimintavuoroalueella.

Eri ampumistyylit olivat yksi hankalimmista asioista toteuttaa. Alun perin ne oli tarkoitus tehdä erillisiksi apufunktioiksi, jotta itse taistelumoottorin rakenne pysyisi selvänä ja muutoksien tekeminen olisi helpompaa. Ampumisen yhteydessä tarvitsisi vain kutsua niitä. Käytännössä tämä kuitenkin tarkoitti laajaa globaalien muuttujien käyttöä. Globaalit muuttujat eroavat normaaleista muuttujista siten, että niitä voi muuttaa myös muualta kuin sen lohkon sisältä, jossa ne määritettiin. Ne olivat siksi tarpeellisia, koska apufunktioiden olisi ollut pakko päästä käsiksi tietoihin, että laskutoimitukset voitiin suorittaa. Vaihtoehto globaaleille muuttujille oli välittää kaikki tarvittavat tiedot apu-

funktioille parametreina, mutta siinä tuli vastaan käytännöllisyyden rajat, sillä parametrien määrät olisivat olleet suuret. Moduulitestien yhteydessä selvisi kuitenkin, ettei globaaleja muuttujia ole hyvä käyttää niiden hankalan hallittavuuden takia. Apufunktioiden koodit liitettiin kiinteästi taistelumoottorin rakenteeseen. Hyvänä puolena tästä saatiin poistettua liiallinen muuttujien lähettäminen edestakaisin. Huonona puolena taistelumoottorin ohjelmakoodin luettavuus ja päivitettävyys kärsii. Jos yhtä asiaa korjaa, pitää se muistaa korjata joka paikasta, jossa entistä yhteistä entistä apufunktion ohjelmapätkää käytetään.

Viimeinen laskettava asia taisteluissa on kierrätetyt resurssit. Jos jommallakummalla puolella on kierrättäjiä elossa, lasketaan niiden kierrättämät resurssit. Työssä toteutettu kierrätys eroaa hieman alkuperäisen pelin toteutuksesta. Alkuperäisesti kierrätys toimi normaalin ampumisen tapaan ja sen myötä oli myös mahdollista, että kierrättäjät kierrättivät kesken taistelun ja sitten tuhoutuivat. Tämä myös tarkoitti sitä, että kierrättäjät joutuivat ampumaan ja että ne suorittivat samat osumislaskelmat kuin muutkin ampuvat alukset. Tätä toteutusta muutettiin irrottamalla kierrätyksen toimintavuoroista ja muuntamalla tapaa, jolla kierrätetyt resurssit lasketaan. Ampumisen sijaan kierrättäjällä on resurssimäärä, jonka verran se pystyy kierrättämään resursseja. Tämä senkin takia, että kierrätys tapahtuu taistelun jälkeen ja elottomia esineitä ei tarvitsisi ampua.

Ennen varsinaista palautuskomentoa tarvitsee tulostaulukot käsitellä. Palautustaulukkoon kootaan molempien puolien menetetyt, jäätyneet ja varastetut alukset. Sen lisäksi varastetut ja tuhotut resurssit sekä planeetan valtaus pitää lisätä myös. Palautustaulukon arvot ovat tarkemmin luettavissa liitteessä 3. Koska kutsuva ohjelma tietää alkutilanteen, näiden tulosten perusteella voidaan laskea kaikki tarvittavat tilanteet taisteluuutiseen, jonka käyttäjät saavat taistelun jälkeen. Viimein taistelumoottori voi palauttaa tuloksen kutsujalle.

5 Testaus

Yksi projektin tärkeimmistä ja samalla vaikeimmista osa-alueista oli testaus. Käyttäjät olettavat, että peli käyttäytyy ohjeissa mainitulla tavalla ja toiminnan pitäisi olla viiveetöntä. Käytännössä tämän kaltainen laaja algoritmi tarvitsee laajaa testausta. Erilaisia käyttötapauksia on äärettömän monta ja niiden keksiminen etukäteen ilman käyttäjätestausta on mahdotonta etenkin, kun palautusarvot voivat olla niin monen arvoisia.
Yksin ohjelmaa tekevälle saattaa tulla kiusaus jättää ohjelmat testaamatta kunnolla, koska taustalla pyörii ajatus, että kyllähän ohjelma toimii halutulla tavalla, koska sen on itse tehnyt. Toisekseen mielessä pyörii testauksen hyödyllisyys, koska testauksen tulokset saattavat monessa tapauksessa jäädä pelkästään omaksi tiedoksi.

Nykyään saatavilla on paljon valmiita testaamistyökaluja, joilla testausta voi helpottaa. Työkalut vaativat kumminkin testausta ja opettelua, joten työkalujen käyttämiseen on olemassa käyttökynnys. Kuitenkin niillä saavutetaan syvempää tietoa ohjelman toiminnasta ja esimerkiksi profiloinnilla sen käyttämistä resursseista. Eli vaikka itsekin saisi tehtyä testausohjelmia, jotka toimivat ja takaavat toimivuuden, on olemassa valmiita työkaluja, joilla saadaan enemmän tietoa. Etenkin suorituskyvyn mittaaminen on hankalaa ja vaatii monen eri komponentin tarkastelua. Näitten ohjelmoiminen aina uudelleen jokaiselle eri ohjelmalle vie aikaa, jonka voisi käyttää itse ohjelman kirjoittamiseen.

Eräs testauksen aikaa vievimmistä vaiheista on sääntöjen selittäminen käyttäjille tai paremminkin testaajille. Suurin osa virheistä, joita funktiosta ilmoitettiin, olivat yksinkertaisia väärinkäsityksiä. Ilmoitukset on kumminkin pakko tutkia askel askeleelta, että onko kyseessä virhe vai normaali toiminta. Taistelumoottorin tapauksessa tapausten läpikäyminen vie aikaa, koska se käytännössä tarkoitti sitä, että tulokset on laskettava käsin ja luvut on todettava oikeiksi.

On helppoa luoda ohjelma, joka käyttäytyy oikein, kun sille annetaan oikeita syötteitä. Ongelmat alkavat sitten, kun sille syötetäänkin jotain aivan muuta. Ilmiselvä tulos tästä voi olla vääränlainen toiminta tai ohjelman kaatuminen. Verkkosovelluksessa on eteenkin tietoturvapuoli erittäin tärkeä, koska väärällä syötteellä voi jopa päästä hallitsemaan palvelinta, jolla sovellus sijaitsee. Silloin ongelmasta tulee laajempi kuin vain omaa sovellusta koskeva, koska palvelimella voi olla muita sovelluksia tai tietoa, johon pääsee

käsiksi. Vaaditaan siis hieman käänteistä ajattelua, jossa lähdetään käyttäjän näkökulmasta. Pitää kyetä miettimään, mitä kaikkea ohjelmalla voidaan ja ei saisi tehdä. Sitten nämä eri tavat pitää estää, joko syötteen seulomisella tai vaikka mahdollisten syötteiden rajaamisella.

Vaikka taistelumoottori ei olekaan altis ulkopuolisille hyökkäyksille tai väärille syötteille samaan tapaan, on tarpeellista mainita asia, koska se täytyy tiedostaa heti alkuunsa parista syystä. Ensimmäisenä syynä on se, että asiat on helpompi tehdä kerralla oikein, kuin että palaisi korjaamaan virheitä myöhemmin. Jo siihen, että tutustuu ohjelmakoodiin ja etsii missä virhe on, menee aikaa ja sitä ennen pitää löytää syy, mistä virhe syntyy. Toisena syynä on yksinkertaisesti se, että tiedostaa siihen kuluvan ajan. Esimerkiksi ohjelma joka summaa kaksi lukua yhteen, on helppo tehdä, mutta kun siihen lisää tarkistukset, sen pituus saattaa helpostikin kaksinkertaistua. Näissä asioissa auttaa tietenkin pitkäkatseisuus kerätä hyödyllisiä funktiota erilliseen paikkaan, josta niitä voi käyttää uudelleen tulevaisuudessa.

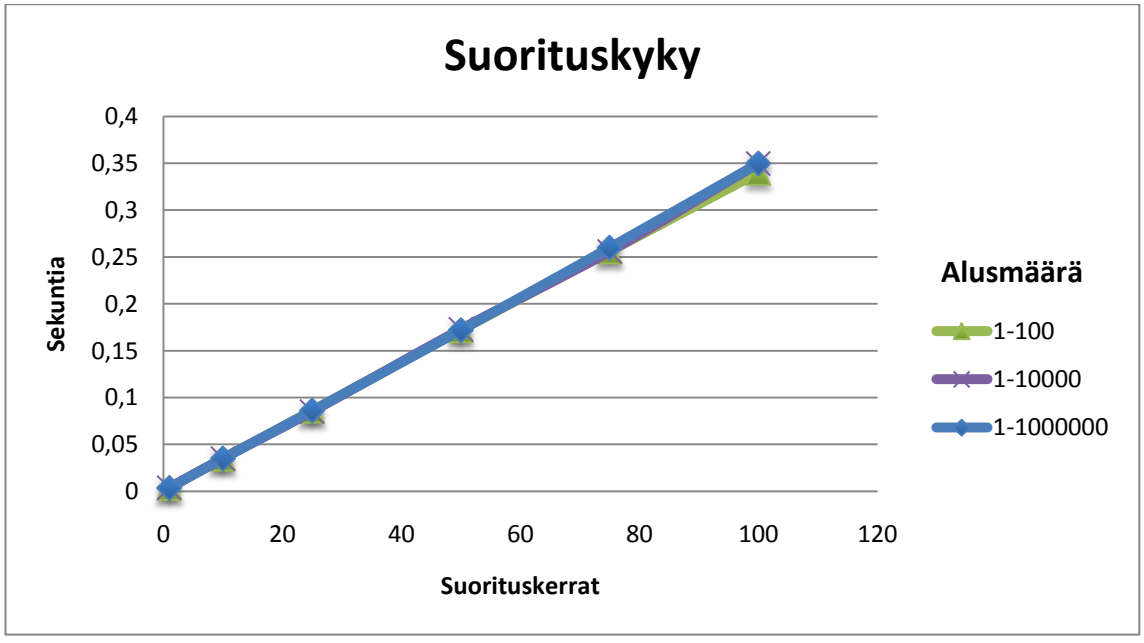
5.1 Kuormitus- ja suorituskykytestaus

Internet-sovelluksen tärkeä osa-alue on suorituskyky ja toimintavarmuus. Testaamiseen luotiin yksinkertainen PHP-ohjelma, joka arpoi satunnaiset arvot halutulta väliltä syötettäväksi taistelumoottorille. Tämä laitettiin silmukkaan, jota ajettiin haluttu määrä ja suoritukseen mennyt aika saatiin palvelimen aikaleiman avulla. Palvelimena toimi Linux-käyttöjärjestelmän päällä oleva Apache2 ja siihen oleva PHP-moduli. Asetukset olivat oletusasetuksia ilman optimointeja. Palvelimen rautapohjana oli Intel Pentium 4 -prosessoriin pohjautuva vanha työpöytäkone.

Aluksi ohjelmaa käytettiin puhtaasti taistelumoottorin rasittamiseen satunaisilla arvoilla ja tilanteilla. Sillä tavoin saatiin hieman kuvaa siitä, löytyikö taistelumoottorista kriittisiä virheitä, jotka pysäyttäisivät sen suorituksen. Löytyneet virheet korjattiin. Tämän testaustavan haittapuolena on se, ettei taistelumoottorin palauttamia tuloksia voinut tarkistaa ja siten myös niitten oikeellisuus jäi selvittämättä. Satunnaisten alusmäärien laskemisesta tulee pientä viivettä suorituskykyyn, mutta se on niin mitätöntä, ettei sitä tarvitse huomioida.

Testissä lähdettiin testaamaan alusten ja suorituskertojen määrän vaikutusta suorituskykyyn. Alkuperäisessä pelissä suorituskyky heikkeni nopeasti, kun alusten määrät nousivat, koska siinä laskettiin joka laukaukselle satunnaisuuksia. Satunnaisuuksien laskeminen taisteluissa, jossa oli miljoonittain aluksia ja jokaisella aluksella on useampia laukauksia, aiheutti eksponentiaalisen laskutoimitusten määrän nousun. Lopputuloksena oli pelin kaatuminen ja taisteluiden jääminen ratkaisematta.

Testitapaukset määriteltiin seuraavasti. Suorituskerroiksi valittiin 1, 10, 25, 50, 75 ja 100, jotta saataisiin kuvattua mahdollista suoritusnopeuden hidastumista ajokertojen määrien kasvaessa. Alusmääriksi valittiin 3 tapausta: 1-100, 1-10000 ja 1-1000000 alusta. Tulokset on piirrettynä kuvassa 12. Tällä tapaan saatiin kuvaajaan myös mukaan alusten määrän lisääntymisen vaikutukset suoritusaikaan. Alarajaksi valittiin yksi alus siksi, että sillä tavoin saatiin mahdollisimman monta alustyyppiä mukaan taisteluun kuvastamaan raskainta laskentatilannetta taistelumoottorille. Jokainen testitapaus ajettiin 5 kertaa ja niistä laskettiin keskiarvo. Tarkempi mittauspöytäkirja löytyy liitteestä 5.



Kuva 13 . Suorituskyky erilaisilla alusmäärillä

Taistelumoottori osoittautui hyvin tehokkaaksi. Yhden taistelun laskemiseen meni noin 0.004 sekuntia eli sekunnissa voisi siis laskea noin 250 taistelua. Teoreettisesti ajateltuna, jos oletetaan palvelimen suoritusrajoituksen olevan vakio 30 sekuntia, voisi sen aikana siis ajaa jopa 7500 taistelua. Käytännön laskeminen kunnon optimoidulla palve-

limella on vielä huomattavasti nopeampaa, mutta näitä eroja ei ruveta arvioimaan, koska erilaisia alustoja ja asetuksia on monia. Toinen huomionarvoinen asia on alusten määrän vaikutus suorituskykyyn. Kuvasta 13 näkee, että suorituskyky pysyy samana, vaikka alusten määrä vaihtelee. Tämä ominaisuus oli tärkeä vaatimus taistelumoottorin suunnittelussa ja testauksesta nähdään, että se on toteutunut. Tuloksen perusteella voidaan olla varmoin ja luottavaisin mielin myös silloin, kun alusten määrät kasvavat.

5.2 Testaus ennalta määrätyillä testitapauksilla ja taistelusimulaattorilla

Testauksen pitää alkaa heti siinä vaiheessa, kun ohjelmakoodia ruvetaan kirjoittamaan. On toki mahdollista kirjoittaa laajempia ohjelmakokonaisuuksia testaamattakin, mutta jos virheitä ilmaantuu, on niiden etsiminen todella hankalaa. Siksi jokainen pieni apufunktio ja osa-alue kannattaa testata erikseen, ennen kuin luottaa siihen, että se toimii halutulla tavalla ja alkaa käyttää sitä osana kokonaisuutta. Tietenkään edes tämä ei takaa sitä, että se toimii varmasti.

Ensimmäiset testit tehtiin pienen PHP-ohjelman avulla. Siinä muokattiin käsin funktiolle syötettäviä arvoja ja tulosteet tulivat yksinkertaisina syöte- ja palautearvojen listauksina. Tämä tapa on kumminkin hyvin työläs ja vaikealukuinen, joten seuraavana vaiheena taistelumoottorille luotiin simulaattorisivu [3], jossa oli alkeellinen käyttöliittymä tietojen syöttämiseen. Nappia painamalla syötetyt arvot lähettiin taistelumoottorille ja ruudulle tulostettiin saatu tulos selkeämmässä taulukkomuodossa. Tulokset myös järjesteltiin eri osa-alueisiin, jolloin niitä oli helpompi tulkita. Tämän lisäksi taistelumoottoriin lisättiin eri kohtiin koodia tulostuksia, jotka kertoivat, mitä se kunakin hetkenä oli tekemässä. Taistelujen laskemista pystyi siis seuraamaan askel askeleelta.

Moduulitestaus on hyvä tapa testata osa-alueiden tai funktioiden toimintaa. Siinä luodaan testimetodeita, jolla kutsutaan testattavia funktioita. Siinä käytännössä tarkkaillaan alkutilanteen ja lopputilanteen eroja eli sitä, onko funktio käyttäytynyt halutulla tavalla. Esimääritettyjen testitapausten käyttäminen muistuttaa hyvin paljon moduulitestausta. Siinä tiedetään haluttu käyttötapaus ja tulos, jonka funktion pitäisi palauttaa. Taistelumoottorin tapauksessa ennalta määrätyt testitapaukset olivat taisteluita, joiden osallistujien alusmäärät ja lopputulos tiedettiin. Käytännössä siis taistelut kirjoitettiin

käsin paperille askel askeleelta. Samat lähtötiedot syötettiin taistelumoottorille ja saatua tulosta verrattiin ennalta laskettuun tulokseen.

Liitteessä 5 on taulukko erilaisista taistelumoottorin testitapauksista. Testitapaukset käsittelevät taistelumoottorin käyttäytymistä ja niissä ei ole keskitytty kertomaan kussakin testissä käytettyjen aluksien määrää, sillä silloin erilaisten testitapausten määrät nousevat huimiksi. Suurimmaksi osaksi eri alusmäärien yhdistelmiä kokeilivat yhteisön pelaajat, mutta he eivät tietenkään dokumentoineet testejään. Yhteisötestaamisen piirteenä on, että vain ongelmatapaukset ilmoitetaan, jolloin toimivat tapaukset jäävät ilmoittamatta.

Ennalta määrätyillä testitapauksilla testaaminen ei kuitenkaan pysty tuottamaan täysin toimivaa tuotetta taistelumoottorin monimutkaisuuden ja eri tapausten määrän vuoksi. Tähän tilanteeseen toi helpotusta yhteisö, jonka pelaajat olivat innokkaita auttamaan ja testaamaan uutta peliin tulevaa taistelua. Simulaattori siis julkaistiin nettisivulla [3] testattavaksi kaikille halukkaille. Tällä tavoin yhteisö saadaan testaamaan ohjelmoijan puolesta ja testitapausten määrä nousee paljon suuremmaksi ja kattavammaksi mitä kukaan yksi ihminen voisi ajatella. Mahdolliset virheet käyttäjät saivat ilmoittaa ennalta määritetyssä muodossa pelin foorumille. Pelaajat eivät usein myöskään miellä testausta työksi, koska se on heille tapa päästä vaikuttamaan ja näkemään etukäteen miten peli tulee käyttäytymään. Se siis kasvattaa samalla yhteisön yhteishenkeä.

Viimeiseksi vaiheeksi jäi taistelumoottorin testaaminen itse pelissä. Tätä varten tehtiin erillinen soviteohjelma tai tulkki, joka tulkitsi sekä käytti taistelumoottoria. Muutamaa pienempää virhettä lukuun ottamatta itse funktio toimi hyvin. Ongelmia sen sijaan aiheutti funktion sitominen pelin tietokantaan ja sen tietotyyppeihin. Yksi PHP:n ominaisuuksista on se, ettei eri muuttujien tyyppejä tarvitse juurikaan määrittää, vaan se itse määrittää ne automaattisesti tapauskohtaisesti. Eri muuttujien tyyppimuunnokset MySQL-tietokannasta aiheuttivat ongelmia, koska pienetkin erot saattoivat rikkoa apufunktioiden toimintaa. Eräässä tapauksessa virheen aiheuttamiseksi riitti se, että tietokannassa sijaitsevien arvojen perässä oli välilyönti.

6 Kokemukset ja jälkiviisaudet

Tässä luvussa keskityn kertomaan hieman henkilökohtaisemmasta näkökulmasta havaintoja ja jälkiviisauksia, jotka sain työn aikana. Lähdin projektiin avoimin mielin oppimaan uutta asiaa ja kehittämään itseäni paremmaksi ohjelmoijaksi. Alkutilanne oli se, ettei minulla ollut käytännön kokemusta laajempien ohjelmien toteuttamisesta tai kehityksen aikana ilmentyvistä ongelmista.

Suurin asia, minkä muuttaisin taistelumoottorissa, olisi olio-ohjelmoinnin käyttö. Pois jättäminen johtui siitä, etten tiennyt sen olevan edes mahdollista vasta kuin jälkikäteen. Etenkin syöte ja palautearvojen käsittelyssä siitä olisi huomattavasti apua. Nykyisellään taistelumoottori ottaa syötteenä ennalta määritetyn taulukon, joka sisältää tiedot hyökkääjästä ja puolustajasta. Jälkiviisaana tiedän, että tämä haittaa koodin modulaarisuutta, dynaamisuutta ja uudelleenkäyttöarvoa. Tietyn arvon täytyy olla tietyssä kohdassa taulukossa, mikä myös hankaloittaa sen käyttöä. Sama kankeus jatkuu myös laajemmin käsittelyyn, sen takia koska tietoa käsitellään taulukkona. Syöte ja palautusarvo olisi ehdottomasti pitänyt toteuttaa olio-ohjelmoinnilla. Olioon olisi voinut paljon dynaamisemmin sijoittaa arvot ja se olisi helpottanut esimerkiksi siirtoa toisiin peleihin. Oliototeutus on tällä hetkellä tärkein toteutettava ominaisuus, jos taistelumoottori tulee käyttöön toisessa pelissä.

Toinen asia minkä toteuttaisin olio-ohjelmoinnilla, olisi alusten ominaisuuksien tallentaminen. Jos ominaisuudet olisivat tallennettu oliossa, olisi niiden käyttö ja muokkaaminen huomattavasti dynaamisempaa. Jotta päivitys olio-ohjelmointiin onnistuisi, tarvitsisi peliäkin ja sen tietokantaa muuttaa, koska siinä on monet alukset kiinteästi sidottu sivuihin. Niiden dynaaminen hakeminen ei siis toimisi ja yhden osan muuttaminen kaataisi muut sitä osaa käyttävät osat, jos kaikkia sivuja ei kävisi läpi. Nykyisessä kiinteässä mallissa suurin etu muodostuu siitä, että se karsii tietokantakyselyjen määrää ja on yksikertainen. Yksinkertaisella tarkoitan sitä, että sitä voisi kuka tahansa ohjelmoinnista tietämätönkin muuntamaan millä tahansa tekstinkäsittelyohjelmalla.

Henkilökohtaisemmalla tasolla kärsivällisyys on tärkeää. Toteuttaessa isoa kokonaisuutta tulee usein seiniä eteen. Niiden yli kiipeäminen tuntuu mahdottomalta ja tuloksena on turhautuminen. Saattaa olla, että palaset erikseen toimivat, mutta yhdessä ne eivät suostu millään toimimaan. Vaikka kuinka yrittäisi lukea työtään läpi kriittisellä silmällä,

vikaa ei löydy. Omassa mielessä näytöllä oleva ohjelmakoodi on juuri niin kuin sen pitääkin olla. Tämän tilanteen nimesin koodisokeudeksi. Paras neuvo, jota voin tilanteeseen antaa on se, että välillä pitää ottaa etäisyyttä ohjelmointiin ja tehdä jotain muuta. Hetken levon ja irtautumisen jälkeen virhe saattaa suorastaan hypätä silmille, kun ohjelmakoodin avaa muokattavaksi seuraavan kerran. Toinen hyvä neuvo on, ettei aina kannata yrittää täysin yksin. Mahdollisuuksien rajoissa kannattaa näyttää ongelmaa ohjelmoinnista ymmärtävälle kaverille. Toiset ihmiset saattava huomata hyvinkin yksinkertaisia ja helppoja tapoja toteuttaa tai korjata samat asiat, jotka itse on vaikeimman kautta tehnyt. He myös näkevät useimmiten virheetkin nopeammin, koska eivät ole juuttuneet samaan mielentilaan, jossa itse on tehnyt ohjelmaa. Nämä neuvot eivät tietenkään päde vain ohjelmointiin, vaan melkeinpä kaikkeen, mikä liittyy uuden asian luomiseen.

Viimeisenä jälkiviisautena on varoitus PHP-kirjaston valmiista funktioista. PHP:hen on saatavilla paljon erilaisia valmiita funktioita ja kirjastoja aivan kuten muissakin ohjelmointikielissä. Näillä pyritään saamaan toistuvien tehtävien ohjelmoinnin määrää vähentymään. Täytyy kumminkin antaa varoituksen sana joidenkin näiden käytöstä. Työtä tehdessä kävi niin, että tietorakenteita käsittelevät valmiit funktiot osoittautuivat epävakaiksi. Luotin siihen, että ne olisivat täysin testattuja ja luotettavia, koska ne oli sisällytetty viralliseen PHP:n funktiokirjastoon [5]. Esimerkiksi funktio, joka etsi taulukon läpi ja ilmoitti, että onko kysyttyä arvoa taulukossa. Käytännössä se siis säästää kirjoittamasta erillistä ohjelmapätkää, joka silmukan avulla kävisi taulukon läpi. Sisällytin sen erään taistelumoottorin apufunktion osaksi sitä sen kummemmin testaamatta. Apufunktio vaikutti toimivan oikein, kun sitä erillisenä osana testattiin.

Myöhemmässä vaiheessa ilmeni toimintavirhe taistelumoottorin osassa, jossa tämä apufunktio oli käytössä. Ongelmaa etsittiin pitkään sen takia, ettei PHP:n kirjastossa ollutta valmista funktiota osattu epäillä. Lopuksi päädyttiin purkamaan apufunktion osiin ja testaamaan jokaisen osa uudelleen, jolloin vika kohdistui juuri tähän mainittuun valmiiseen funktioon. Se palautti satunnaisesti väärän tuloksen. Arvo saattoi olla taulukossa, mutta funktio palautti vastauksena, ettei sitä löytynyt. Toimintavirhe korjaantui sen jälkeen, kun valmiin funktion toiminta ohjelmointiin itse ja se sisällytettiin apufunktioon. En toki tarkoita, että jokainen PHP:n apufunktio saattaisi olla samaan tapaan epävakaa, mutta kehotan käyttämään niitä terveen vainoharhaisuuden kanssa. Kun

uuden valmiin funktion ottaa käyttöön ensimmäistä kertaa, kannattaa sen toiminta testata kunnolla, koska jos niissä ilmenee ongelmia, jälkikäteen vian etsiminen vie paljon aikaa.

7 Yhteenveto

Työssä lähdettiin tekemään taistelumoottoria peliin, jonka alkuperäisesti oli toiminut 2000-luvun alussa. Peliä heräteltiin henkiin tekemällä se alusta asti uudelleen vapaaehtoisvoimin kansainvälisellä ryhmällä. Pohjalla oli vain vanhan pelin manuaali, jossa taistelun kulku oli selitetty epämääräisesti. Sen ja vanhojen pelaajien muistojen mukaan taistelumoottori saatiin jäljittelemään vanhan pelin toimintaa ja ulkonäköä hyvin.

Työn valmistuttua saattoi helposti tajuta hyvän suunnittelun hyödyn. Hyvin suunniteltu työ välttyy ajan tuhlaamiselta esimerkiksi tapauksissa, jossa joudutaan tekemään kokonainen alue ohjelmasta uudestaan, koska alkuperäiset lähtökohdat oli suunniteltu väärin. Myös testauksen tärkeys korostuu samasta syystä. Ohjelmat on syytä tehdä kerralla niin valmiiksi kuin voi. Jos tyytyy vain testaamaan sen, että ohjelma toimii oikeissa olosuhteissa halutulla tavalla, jää useimmiten virhetilanteet huomaamatta. Kun ongelmia ilmenee ennemmin tai myöhemmin, menee niiden etsimiseen pitkä aika. Yritysmaailmassa aika on kallis hyödyke.

Itse taistelumoottori on monen muun ohjelman tavoin jatkanut kehitystään ensimmäisestä versiostaan ja on saanut uusia ominaisuuksia. Pelkästään sen peliin liittäminen antoi monia uusia ideoita siitä, miten jotkut asiat olisi voitu tehdä toimivammin. Taistelumoottori on toiminut virheettä ja tehokkaasti siitä asti, kun se EmpireQuest-peliin liitettiin. Projektin lopputulos oli siis onnistunut ja hyvä. Työstä kertyi paljon jälkiviisautta, jonka voimin jaksaa ja pystyy ajattelemaan tulevia projekteja aikuistuneemmalta kantilta. Parasta opituissa asioissa on, etteivät ne tee asioita vaikeammaksi toteuttaa vaan juuri päinvastoin. Aikaa säästyy, kun ei toista samoja virheitä.

Nähtäväksi jää, missä muodossa taistelumoottori ja peli, jota varten se tehtiin, jäävät elämään. Vapaaehtoisprojektina siinä on paljon ongelmia, jotka pitäisi saada selvitettyä eteenkin vastuun jakamisesta. Voi sanoa, että pelin pyörittämisestä kertyy kokemusta,

jonka päältä on helpompi lähteä rakentamaan uusia projekteja. Alusta asti taustalla on ollut haaveena kehittää myös oma peli, johon voisi toteuttaa enemmän uusia ideoita. Vahva yhteisö on pakollinen ja yksin tai ilman tukea en enää lähtisi luomaan peliä. Tässä projektissa jouduttiin pidättäytymään vanhoissa säännöissä, koska kyse oli vanhan pelin henkiinherättämisestä eikä uudelleen keksimisestä.

Lähteet

1 EmpireQuest-peli. Verkkosivu. <http://www.empirequests.com/>.

2 Alkuperäisen EmpireQuest-pelin manuaali. Verkkosivu. <http://secondconflict.com/eq/manual/>. Julkaisuvuosi 2001.

3 Taistelumoottoria käyttävä simulaattori. Verkkosivu. <http://empirequests.com/simulator/>. Päivitetty ja testattu 1.11.2010.

4 Wikipedia - Batch processing. Verkkosivu. <http://en.wikipedia.org/wiki/Batch_processing>. Päivitetty 30.5.2011.

5 PHP.net. Hypertext preprocessor documentation. Verkkosivu. <http://www.php.net/docs.php>.

6 Wikipedia – MySQL. Verkkosivu. <http://en.wikipedia.org/wiki/MySQL>. Päivitetty 29.5.2011.

EmpireQuestin-alusten käyttäytyminen taisteluissa.

Ohessa on kopio Internet-arkistosta löytyneestä EmpireQuestin alkuperäisen manuaalin selitys taistelujen etenemisestä:

General Behaviour of Ships

Class	General	Successful Shot Formula
Combat Ships, Ship Boarder, and Stinger	Targets ships in its target class, if it can't find any, it will target other ships randomly. If there are no ships it will then target defence.	15% + (accuracy - (agility of target / 2)) %
Troop Carrier and Troop Transport	If the target has 30% or less of the attacker's score, the attacker has 5 colonies already, or there are Shield Generators then they refuse to take it. If they refuse to the planet they will commence to destroy mines and probes if there are no annexers there.	10% + (accuracy) %
Resource Frighter	Takes Tellerium and Krypton from the current tick's production.	10% + (accuracy - 17.5%) %
Pirate	Functions the same as when a Combat Ship attacks ODG.	15% + (accuracy - 10%) %
Annexer	If there are Shield Generators they refuse to steal.	10% + (accuracy - 17.5%) %
Salvagers	Salvages 50% of the value of a ship.	10% + (accuracy - enemy agility) % If the salvager is an attacker then the enemy's agility is 25% otherwise it's 15%.

Note:

- When Combat Ships and Pirates are fighting ODG, the ODG's agility is 10%.
- Agility is not divided by 2 when Combat Ships fight ODG.

Combat (outdated description of the ticker)

Combat in EmpireQuest is carried out whenever there are foreign fleets at a planet (enemy or allies). The ticker organizes each fleet into fleet slots, each fleet slots consists of the following:

- base planet
- ship type
- init
- quantity
- armour values
- targeted (# of times ships in this fleet slot have been targeted for killing)
- side (attackers, defenders)

The defences are also organized in a similar structure, except base planet does not exist, and ship type becomes defence type.

The fleets and defences (ODG) present at the combat location are organized according to their init order (1...15). For each init level the following is looped over until it is false:

- checks that there is defence slots or fleet slots to be processed
    - picks whether to process defence or fleet (in a no choice situation, where only defence or fleet slots are left it picks that)
        - **if it has picked to process a fleet then the following is carried out:**
        - looks up what side (attackers,defenders) it picked last time (first time defenders are picked)
        - fleet slot sides are picked in groups of 2

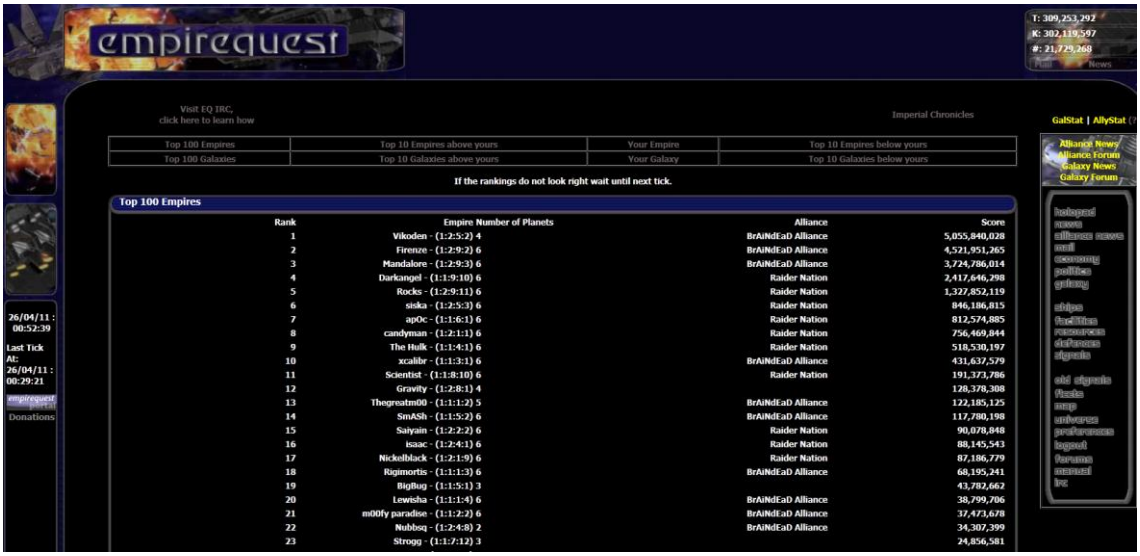
- depending on what it picked last time and if it was the 1st or 2nd group of the side, it will pick attackers or defenders
- after the side has been decided it will try to find a fleet slot matching that criteria, if it cannot it will just pick a fleet slot that has not already been calculated and ignore what side it is on
    - **once a fleet slot has been picked it checks for the following conditions to be true before it proceeds:**
    - Planet Capturing ships on the defenders side are skipped in the calculation
    - If the number of ships lost equals the quantity then it is skipped
    - A gun pool (number of ships * number of guns) is calculated, only non-disabled and alive ships are taken into account
    - The following actions are repeated until the gun pool is equal to zero
    - **If the ship type is a mine and probe stealer than the following is carried out:**
        - A shoot chance (10 + ship accuracy - planet agility / 2) is calculated, the planet agility is 35%
        - If the shot is successful it is picked whether mines or probes should be stolen, else skip to end
        - More than 10% of all mines and probes on that planet cannot be stolen, therefore if there is not enough of the chosen type it will choose the other if that has enough, if both do not have enough, then the mine and probe stealer will skip this gun shot.
        - **end, regardless if shot was successful or not:** Gun pool is updated to reflect the shot
    - **If the ship type is a resource stealer than the following is carried out (Note that it can steal from all the resources produced by the target empire the current tick, not just from the victim planet):**
        - Makes sure they are on the attackers side, if they are not, then this fleet slot is marked as calculated and skipped
        - A shoot chance (10 + ship accuracy - planet agility / 2) is calculated, the planet agility is 35%
        - If the shot is successful it is picked whether tellerium or krypton should be stolen, else skip to end
        - The chosen resource type is stolen in increments of 610 for each shot, if there is enough left.
        - **end, regardless if shot was successful or not:** Gun pool is updated to reflect the shot
    - **If the ship type is a ship boarder:**
        - A target fleet is tried to be found searching the fleet slots that are on the opposite of the current one:
            - **Finding Target Fleet**
            - If the target fleet has lost as many ships as it has then it is skipped
            - If the current fleet slot is an Ion ship then it makes sure all the ships have not been blocked already, else the target fleet is skipped
            - If the current fleet slot is a Ship Boarder type then it makes sure there are ships that have been disabled by an Ion Ships or Ion Defence, else the target fleet is skipped
            - If the current fleet slot's target class matches the target fleet slot's ship class then it will consider it as the target fleet to fire at
            - Each fleet slot keeps a count of how many types it has been targeted, the targeting code tries to find a target fleet with the lowest target count as to even out losses among the same side
            - If the current fleet slot could not find a viable target fleet, and the current fleet target ship class is not set to all, then it is set to all, and another search is performed
        - If a target fleet cannot be found then the current fleet is set as calculated and skipped

        - All the ships in the target fleet are searched to find one that is disabled for the ship boarder to take over
        - Once one has been found a shoot chance is calculated (10 + current fleet slot ship accuracy)
        - If the shot is successful the targetship is moved over to the side of the current fleet slot, as part of the fleet of the current fleet slot
        - **end, regardless if shot was successful or not:** Gun pool is updated to reflect the shot
- **If the ship type is an ion ship:**
    - A target fleet is tried to be found searching the fleet slots that are on the opposite of the current one:
        - **Finding Target Fleet (see ship boarder)**
    - If a target fleet cannot be found then the current fleet is set as calculated and skipped
    - All the ships in the target fleet are searched to find one that is not disabled for the ion ship to disable
    - Once one has been found a shoot chance is calculated (15 + current fleet slot ship accuracy - target fleet ship agility / 2)
    - If the shot is successful the target ship is disabled
    - **end, regardless if shot was successful or not:** Gun pool is updated to reflect the shot
- **If the ship type is a combat ship:**
    - A target fleet is tried to be found searching the fleet slots that are on the opposite of the current one:
        - **Finding Target Fleet (see ship boarder)**
    - If a target fleet cannot be found then the current fleet targets defence, go to **kill defence**
    - All the ships in the target fleet are searched to find one that is not dead for the combat ship to kill
    - Once one has been found a shoot chance is calculated (15 + current fleet slot ship accuracy - target fleet ship agility /2) and the number of shots required (current fleet slot ship gun power / armour of target ship), if the number of required shots is greater than gun pool then the required shots are set to equal the gun pool
    - If the shot is successful the damage (required shots * current fleet ship gun power) is subtracted from the target fleet's armour
    - **end, regardless if shot was successful or not:** Gun pool is updated to reflect the shot
    - **kill defence**
    - A target defence is tried to be found searching the defence slots:
        - If the target defence has lost as many defences as it has then it is skipped
        - Each defence slot keeps a count of how many types it has been targeted, the targeting code tries to find a target defence with the lowest target count as to even out losses among the defences
        - If the current fleet slot could not find a viable target defence, and the current fleet target ship class is not set to all, then it is set to all, and another search is performed (yes, defences have ship classes)
    - All the defences in the denfece slot are searched to find one that is not dead for the combat ship to kill
    - Once one has been found a shoot chance is calculated (15 + current fleet slot ship accuracy - target fleet ship agility /2) and the number of shots required (current fleet slot ship gun power / armour of target ship), if the number of required shots is greater than gun pool then the required shots are set to equal the gun pool
    - If the shot is successful the damage (required shots * current fleet ship gun power) is subtracted from the target fleet's armour
    - **end, regardless if shot was successful or not:** Gun pool is updated to reflect the shot

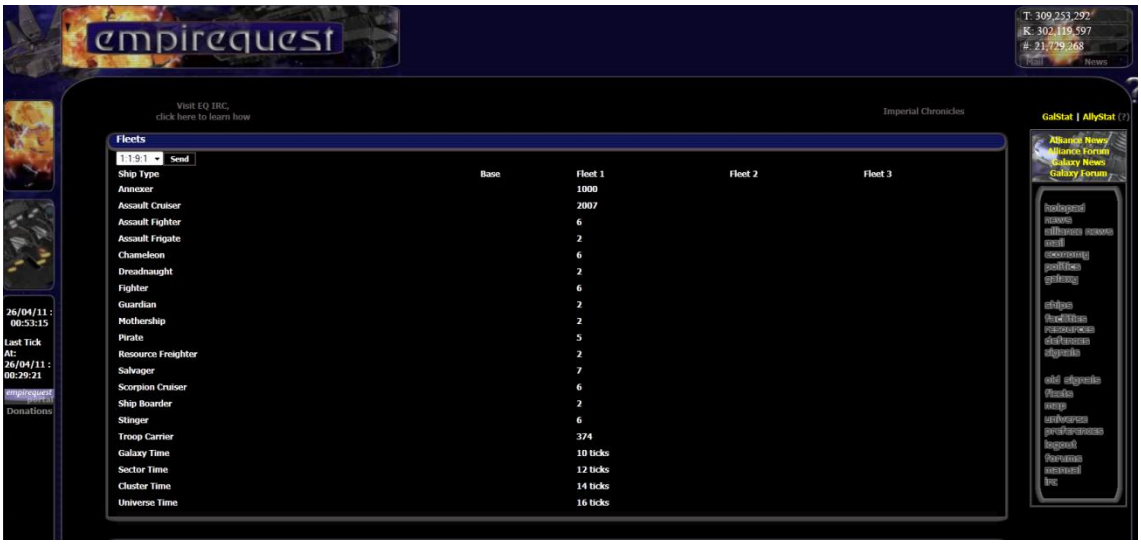
- **If the ship type is a pirate:**
  - **see kill defence (combat ship)**
- **If the ship type is a capture taking ship:**
  - A shoot chance is calculated (10 + fleet slot ship accuracy)
  - If the shot was successful then it is seen if the planet is a home-world or if it has been already taken over in this combat, if one of those is true (if none of them are true then it tries to take over the planet, first it makes sure the player does not have 6 already, if it does not, then the owner of the current fleet slot takes over the pla-net), if the planet has a owner (not new, original) then mines and probes are set to be killed, else if it is an unsettled planet it does not destroy them.
  - **If the mines and probes are set to be killed then the following is carried out:**
    - If there are any annexers present on the attackers side then it does not kill any mines or probes
    - It chooses whether to steal mines or probes, if however no mines are left, then probes are chosen and vice versa
    - The planet mine and probe counters and stolen counters
- **If however it has picked to process a defence slot then the following is carried out:**
- Finds a defence slot that has not been calculated yet
- see Ion Ship for Ion Defences and Combat Ship for all other types of defences, since they behave like the ships, also the chance shoot for defence is 15 + accuracy - agility / 2
- Orbital Mines are liked regular combat ships except they kill any target instantly if the shot succeeds and it is only one gun shot that is decremented

Kuvia EmpireQuest-pelistä

Ohessa on kuvankaappauksia EmpireQuest-pelistä. Ulkoasu on sama uudelleen tehdyssä pelissä, sekä vanhassa jonka perusteella ulkonäkö luotiin.

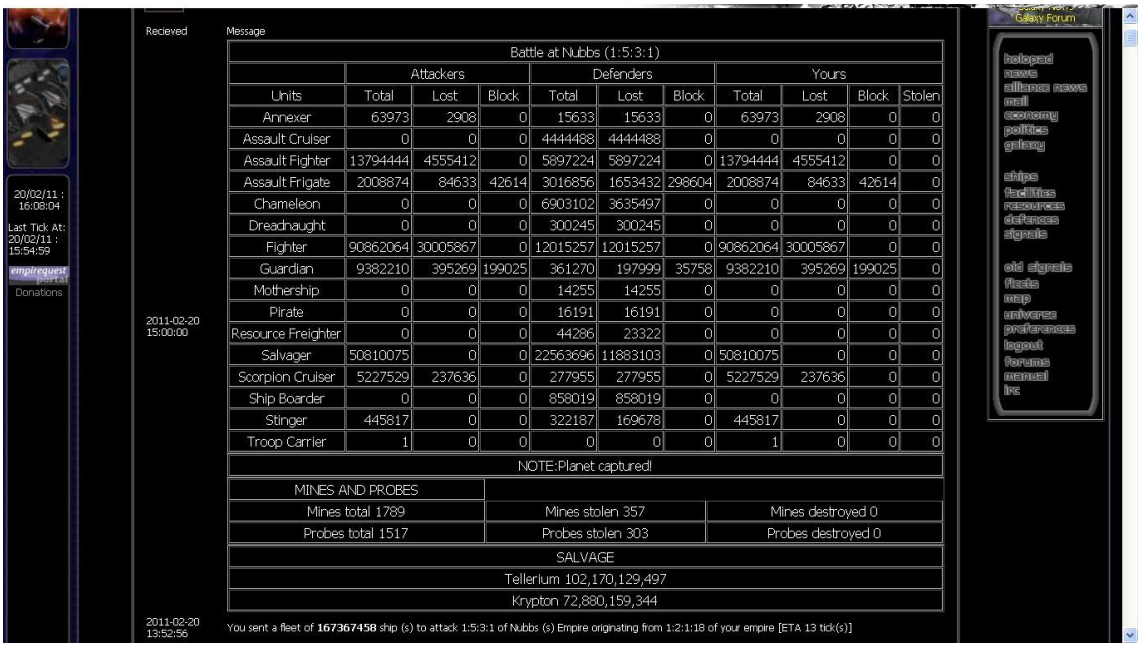


Kuva 1. Universumin-selaamissivu



Kuva 2. Laivastojen kontrollointisivu.

Kuva 3. Ships-sivu



Battle at Nubbs (1:5:3:1)

	Attackers			Defenders			Yours			
Units	Total	Lost	Block	Total	Lost	Block	Total	Lost	Block	Stolen
Annexer	63973	2908	0	15633	15633	0	63973	2908	0	0
Assault Cruiser	0	0	0	4444488	4444488	0	0	0	0	0
Assault Fighter	13794444	4555412	0	5897224	5897224	0	13794444	4555412	0	0
Assault Frigate	2008874	84633	42614	3016856	1653432	298604	2008874	84633	42614	0
Chameleon	0	0	0	6903102	3635497	0	0	0	0	0
Dreadnaught	0	0	0	300245	300245	0	0	0	0	0
Fighter	90862064	30005867	0	12015257	12015257	0	90862064	30005867	0	0
Guardian	9382210	395269	199025	361270	197999	35758	9382210	395269	199025	0
Mothership	0	0	0	14255	14255	0	0	0	0	0
Pirate	0	0	0	16191	16191	0	0	0	0	0
Resource Freighter	0	0	0	44286	23322	0	0	0	0	0
Salvager	50810075	0	0	22563696	11883103	0	50810075	0	0	0
Scorpion Cruiser	5227529	237636	0	277955	277955	0	5227529	237636	0	0
Ship Boarder	0	0	0	858019	858019	0	0	0	0	0
Stinger	445817	0	0	322187	169678	0	445817	0	0	0
Troop Carrier	1	0	0	0	0	0	1	0	0	0

NOTE:Planet captured!

MINES AND PROBES		
Mines total 1789	Mines stolen 357	Mines destroyed 0
Probes total 1517	Probes stolen 303	Probes destroyed 0

SALVAGE

Tellerium 102,170,129,497

Krypton 72,880,159,344

Kuva 4. Esimerkki taistelu-uutisesta

Taistelumoottorin syöte- ja palautustaulukoiden määrätty sisältö

Taulukko 1. Syötetaulukon ennalta määrätty sisältö

Indeksi	Sisältö
0	Tyhjä, koska alusten ID-luvut alkaa 1:stä
1—17	Hyökkäävä laivasto: Jokaista ID-lukua vastaava lukumäärä
18	Hyökkääjän pistemäärä
19	Hyökkääjän planeettojen lukumäärä
20—42	Puolustajan laivasto + planeetan puolustusjärjestelmät
43	Puolustajan 1. resurssimäärä
44	Puolustajan 2. resurssimäärä
45	Puolustajan kaivokset (tuottaa resurssia 1)
46	Puolustajan luotaimet (tuottaa resurssia 2)
47	Puolustajan pistemäärä

Taulukko 2. Palautustaulukon ennalta määrätty sisältö

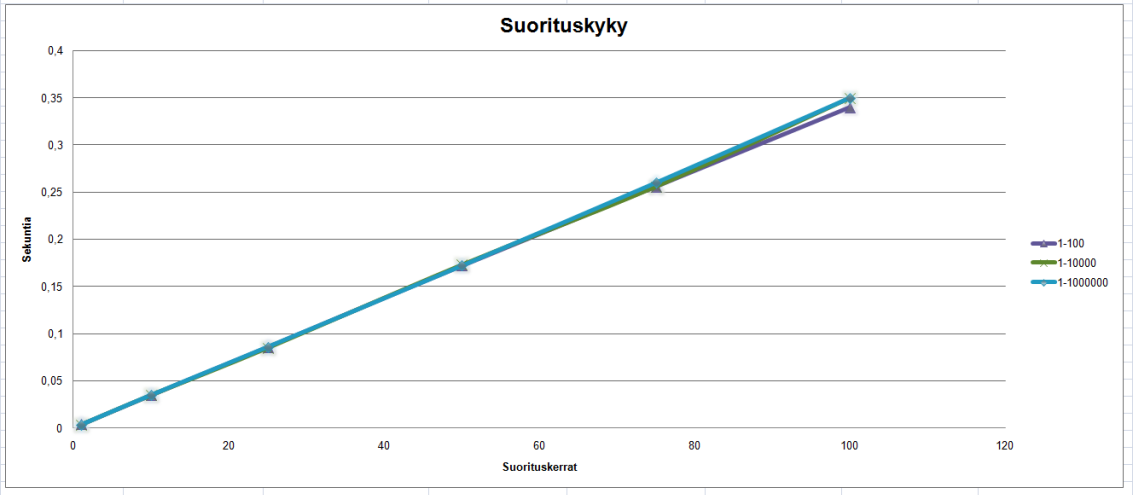
Indeksi	Sisältö
0	Tyhjä
1-17	Hyökkääjän menetetyt alukset
18-34	Hyökkääjän jäädytetyt alukset
35-51	Hyökkääjältä varastetut alukset
52-74	Puolustajan menetetyt alukset
75-97	Puolustajan jäädytetyt alukset
98-120	Puolustajalta varastetut alukset
121	Hyökkääjän kierrätetty resurssi 1
122	Hyökkääjän kierrätetty resurssi 2
123	Puolustuksen kierrätetty resurssi 1
124	Puolustuksen kierrätetty resurssi 2
125	Varastetut kaivokset
126	Varastetut luotaimet
127	Tuhotut kaivokset
128	Tuhotut luotaimet
129	Resurssia 1 varastettu
130	Resurssia 2 varastettu
131	Planeetan valtaus on/off-lippu

Suorituskykytestauksen mittauspöytäkirja

Alusten määrä		Suoritusaika sekunteina					
1-100	Taisteluiden määrä	1. suorituskerta	2.	3.	4.	5.	Keskiarvo
	1	0.0039222240447998	0.003931999206543	0.0039119720458984	0.0037760734558105	0.0040009021759033	**0.003908634185791**
	10	0.034275054931641	0.034242868423462	0.037537097930908	0.034386873245239	0.034404993057251	**0.0349693775177**
	25	0.086385011672974	0.085638999938965	0.08576488494873	0.086160182952881	0.087615013122559	**0.086312818527222**
	50	0.17219805717468	0.17071485519409	0.17441606521606	0.17162322998047	0.17241907119751	**0.17227425575256**
	75	0.26078104972839	0.25388407707214	0.2525520324707	0.25783109664917	0.25620198249817	**0.25625004768371**
	100	0.34448099136353	0.33921480178833	0.33933115005493	0.33838701248169	0.3396589756012	**0.34021458625794**
Alusten määrä							
1-10000	Taisteluiden määrä	1. suorituskerta	2.	3.	4.	5.	
	1	0.0036501884460449	0.0035929679870605	0.0042691230773926	0.0036280155181885	0.003532886505127	**0.003734636306763**
	10	0.035178899765015	0.033772945404053	0.035454988479614	0.034590005874634	0.034971952438354	**0.034793758392334**
	25	0.084848807678222	0.085781097412109	0.085787057876587	0.086090087890625	0.084655046463013	**0.085432434082031**
	50	0.17277503013611	0.17293906211853	0.1726450920105	0.17396903038025	0.17169189453125	**0.17280402183533**
	75	0.25473618507385	0.25576496124268	0.25494694709778	0.25684785842896	0.257080078125	**0.25587520599365**
	100	0.340744972229	0.35017085075378	0.34263610839844	0.35935807228088	0.34122109413147	**0.34682621955871**
Alusten määrä							
1-1000000	Taisteluiden määrä	1. suorituskerta	2.	3.	4.	5.	
	1	0.0037119388580322	0.003615140914917	0.0036649703979492	0.0035400390625	0.0036909580230713	**0.003644609451294**
	10	0.034434080123901	0.033936023712158	0.035730838775635	0.035274982452393	0.03525090217590 3	**0.034925365447998**
	25	0.084816217422485	0.085084915161133	0.086177110671997	0.086230993270874	0.085890054702759	**0.08563985824585**
	50	0.17026495933533	0.17192006111145	0.17393112182617	0.17244410514832	0.17358088493347	**0.17242822647095**
	75	0.25817513465881	0.26356983184814	0.25914812088013	0.25834894180298	0.25886487960815	**0.25962138175964**
	100	0.34308791160583	0.35065579414368	0.34515309333801	0.34549784660339	0.36061191558838	**0.34900131225586**

Keskiarvot taulukkona						
	1	10	25	50	75	100
1-100	0.003908634185791	0.0349693775177	0.086312818527222	0.17227425575256	0.25625004768371	0.34021458625794
1-10000	0.003734636306763	0.034793758392334	0.085432434082031	0.17280402183533	0.25587520599365	0.34682621955871
1-1000000	0.003644609451294	0.034925365447998	0.08563985824585	0.17242822647095	0.25962138175964	0.34900131225586

Pyöristetyt arvot kuvaajaan						
Alusmäärä	1	10	25	50	75	100
1-100	0,0039	0,035	0,086	0,172	0,256	0,34
1-10000	0,0037	0,035	0,085	0,173	0,256	0,35
1-1000000	0,0036	0,035	0,086	0,172	0,26	0,35



Kuva 1. Mittauspöytäkirja

Erilaisia testitapauksia

Taulukko 1. Taistelumoottorin testitapaukset

Testitapaus	**Haluttu taistelumoottorin toiminta**	**Tulos**
Virheellinen syöte	Virhekoodin palautus	OK
Kaikkien toimintavuorojen läpikäynti	Taistelumoottori käy koko toimintavuorojen alueen läpi silmukassa.	OK
Laukausvarantoa on jäljellä, mutta kohteet loppuvat	Taistelumoottori lopettaa taistelun jos vastapuolelta loppuu alukset, järjestää tuloksen ja palauttaa sen. Erikoistarkkailussa loputtomat silmukat ja niiden poistaminen.	OK
Rakenne tunnistaa ampuessa kummalla puolella on ampuvaa alusta.	Taistelumoottori tarkistaa kunkin toimintavuoron aluksen kohdalla onko niitä kummallakaan osapuolella ja ampuu sen tiedon mukaan.	OK
Normaali ampuminen	Aluksien aseiden määrä ja tuhovoima lasketaan oikein. Taistelumoottori ampuu aseet loppuun.	OK
Laukauksien osumistarkkuuden laskeminen	Taistelumoottori hakee oikeat arvot ja laskee aluksille oikeat osumistarkkuudet. Myös satunnaisuus (+-5%) lisätään tarkkuuteen.	OK
Kohdealuksien ketteryyden oikeus	Kohdealusten ketteryys haetaan oikein ja siihen lisätään satunnaisuus (+-5%).	OK
Tuhoutuneiden alusten määrän laskeminen	Kohdealuksia tuhoutuu oikea määrä.	OK
Tuhojen siirtäminen toimintavuorojen välissä	Tuhotut alukset siirretään toimintavuorojen välissä väliaikaisista taulukoista oikeisiin taulukkoihin.	OK
Ampumisjärjestys	Alukset ampuvat omalla toimintavuorollaan. Toimintavuorojen välissä tuhotut alukset poistetaan ampuvista aluksista.	OK
Jäädytys	Jäädytys toimii ja jäädytetyt alukset eivät enää toimi.	OK
Varastus	Jäädytetyt alukset pystytään varastamaan onnistuneesti.	OK
Puolustuksen tuhoajan toiminta	Puolustuksien tuhoaja tuhoaa vain puolustusrakennuksia, eikä muita aluksia. Alukset eivät myöskään saa toimia puolustajan puolella ollessaan taisteluissa.	OK
Miinat	Tuhoavat vain pääkohteensa aluksia ja tuhoutuvat itse samalla. Eivät voi olla hyökkäävässä laivastossa.	OK
Resurssien varastus	Resurssien varastajat vievät resursseja puolustajalta.	OK
Miehistönkuljetusalukset	Valtaavat planeettoja hyväksytyissä tapauksissa (kohde yli 30% hyökkääjän pistemäärästä ja kilpiä ei ole) ja tarvittaessa tuhoavat kohteen resurssikenttiä.	OK
Puolustusrakennus: Kilven toiminta	Estävät valtaukset ja resurssikenttien tuhoamisen.	OK
Tuhottujen aluksien kierrätys resursseiksi taistelun jälkeen.	Vain tuhotut alukset lasketaan mahdolliseksi kierrätettäväksi materiaaliksi. Maksimimäärä toimii määritetyllä tavalla (50% kaikkien tuhottujen alusten rakennuskustannuksista).	OK
Palautustaulukon muoto	Palauttaa taulukon jossa arvot ovat oikealla paikallaan.	OK