Miro Kokkonen

Roguelike-pelin kehitys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

5.5.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Miro Kokkonen Roguelike-pelin kehitys 46 sivua 5.5.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Lehtori Miikka Mäki-Uuro

Insinöörityön tavoitteena oli toteuttaa helposti laajennettava roguelike-peli, joka sisältää proseduraalisen sisällön luomisen ja tekoälyn ohjaamia objekteja. Peli toteutettiin C++-ohjelmointikielellä uusimman C++14-standardin mukaisesti ja kehityksessä käytettiin apuna libtcod- eli ”The Doryen Library” -kirjastoa. Pelin moottorista pyrittiin tekemään suorituskykyinen.

Työssä toteutetun pelin sisältö luodaan proseduraalisesti. Määritellyt muuttujat vaikuttavat generoinnin tulokseen, jolloin sisältö on pseudo-satunnaisesti luotua. Pelin alueet generoidaan hyödyntäen toteutusta varten kehitettyjä algoritmeja ja A*- ja flood fill -algoritmien räätälöityjä toteutuksia. Peliin myös luotiin sen peliobjektihierarkian kattava harvinaisuus-järjestelmä, joka vaikuttaa objektien ominaisuuksiin ja niiden esiintymistiheyteen pelin simulaatiossa. Pelin objekteja ohjaamaan toteutettiin tilakoneperiaatteella toimiva tekoäly, joka käyttää A*- ja ”ray casting” -algoritmeja päätösten teossa. Lisäksi peliin toteutettiin näppäinkomennoilla ohjattava käyttöliittymä ja partikkeliefektit.

Libtcod-kirjastosta käytettyjä ominaisuuksia, jotka työn toteutuksen kannalta havaittiin puutteellisiksi, korvattiin työssä toteutetuilla sopivammilla ratkaisuilla. Toteutuksessa jatkokehitystä vaativiksi osa-alueiksi osoittautuivat käyttöliittymän toteutus ja harvinaisuus-järjestelmän liitos pelin objekteihin.

Pelistä saatiin kehitettyä versio, johon on luotu suppea sisältö esittämään pelimoottorin toiminnallisuutta. Toteutettu moottori on käyttökelpoinen ja suorituskykyinen pelin jatkokehitykseen, ja sen osia on mahdollista käyttää myös erikseen uusissa projekteissa.

Avainsanat	roguelike, C++, proseduraalinen, algoritmit, A*, tekoäly, peli, libtcod

Author	Miro Kokkonen
Title	Development of a roguelike game
Number of Pages	46 pages
Date	5 May 2016
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor	Miikka Mäki-Uuro, Senior Lecturer

The purpose of this final year project was to create an easily expandable roguelike game that contains procedural generated content and objects that are controlled by an artificial intelligence. The game was developed using the C++ programming language by following the latest C++14 standard and utilizing the libtcod library that is also known as "The Doryen Library". The developed game engine was aimed to perform well.

The content of the developed game is created pseudo-procedurally guided by defined variables. The game areas are generated utilizing custom implementations of A* and flood fill algorithms and custom algorithms that were created as part of this project. A rarity system covering the entire game object hierarchy was implemented in the game. The system affects objects properties and their prevalence in the game simulation. An artificial intelligence functioning as a state machine was developed to control the game objects. It utilizes A* and ray casting algorithms in its reasoning. Also a user interface controlled with the keyboard and particle effects were created for the game.

Features used from the libtcod library that were perceived as insufficient for the implementation were replaced by more suitable solutions that were developed for the project. It was discovered that further development would be required in the implementation of the user interface and the integration of the rarity system.

A version of the game that contains concise content only to demonstrate the functionality of the game engine was successfully created. The created game engine is utilizable and performs well enough for developing the game further and its components are also usable on their own in new projects.

Keywords	roguelike, C++, procedural, algorithms, A*, artificial intelligence, game, libtcod

Sisällys

1 Johdanto

Insinöörityön tarkoituksena on toteuttaa helposti laajennettava ja suorituskykyinen roguelike-peli C++-ohjelmointikielen moderneja ominaisuuksia hyödyntäen ja käyttäen apuna libtcod-kirjastoa [1]. Insinöörityön avulla on tarkoitus oppia C++-ohjelmointikieltä ja omaksua pelien kehityksessä käytettyjä menetelmiä.

Roguelike-pelit ovat roolipelien alaluokka ja tyypillisesti ne sijoittuvat fantasiamaailmoihin. Niille tunnusomaisia piirteitä ovat vuoropohjaisuus ja satunnaisuuden suuri vaikutus pelin kulkuun. Yleensä niistä myös puuttuu tallennusmahdollisuus ja pelialueet ovat proseduraalisesti generoituja, joten jokainen pelikerta on erilainen [2].

Roguelike-genren nimi tulee Rogue-nimisestä 1980-luvulla kehitetystä roolipelistä, joka oli silloin opiskelijoiden suosiossa koulujen Unix-järjestelmissä [3; 4]. Peli pohjautui sitä edeltäneisiin tietokone- ja pöytäroolipeleihin, mutta sitä pidetään kulmakivenä, johon sitä seuranneet genreen kuuluvat pelit, kuten NetHack, Moria ja Angband ovat vahvasti pohjautuneet [3; 5; 6].

Vaikka genreen kuuluvien pelien teko on nykyäänkin vahvasti harrastelijapohjaista se on kuitenkin 2000-luvun jälkeen inspiroinut myös monia taloudellisesti merkittäviä pelejä kuten esimerkiksi Diablo-pelisarjaa [3; 5].

Roguelike-pelien toteutuksen haasteellisuuteen vaikuttaa niissä käytettyjen algoritmien monimutkaisuus ja pelien laajuus. Nykyään Roguelike-pelin tekeminen on kuitenkin huomattavasti helpompaa kuin esimerkiksi 1980-luvulla, koska kehittäjien on mahdollista hyödyntää valmiita kirjastoja ja lähdemateriaalia ja muiden kehittäjien lähdekoodia on helpommin saatavilla internetistä. Myös ohjelmointikielet ja -työkalut ovat kehittyneempiä ja tietokoneiden suorituskyky ei ole enää merkittävästi rajoittava tekijä.

2 Teknologiat

2.1 The Doryen Library -kirjasto

The Doryen Library, lyhyemmin libtcod, on erityisesti Roguelike-pelien kehitykseen kehitetty kirjasto. Se sisältää Roguelike-pelien toteutusta nopeuttavia työkaluja, kuten yleisesti käytettyjä algoritmeja polun etsintään, näkökentän laskentaan ja korkeuskarttojen generointiin. Lisäksi sen avulla voidaan luoda SDL:n tai OpenGL:n kautta laitteistokiihdytetty konsoli, joka piirron lisäksi käsittelee hiiren ja näppäimistön komennot.

Insinöörityössä hyödynnettiin kirjaston avulla luotua konsolia sekä satunnaislukugeneraattoria ja viivan piirron algoritmia. Toteutuksen alussa hyödynnettiin myös muita kirjaston tarjoamia algoritmeja polunetsintään ja näkökentän laskentaan. Myöhemmissä vaiheissa näiden algoritmien suorituskyvyn jäädessä heikoiksi ja niiden tarjoamat ominaisuudet puutteellisiksi, ne korvattiin omilla vastaavilla nopeammilla ja geneerisemmillä algoritmeilla, joihin halutut ominaisuudet saatiin toteutettua. Kirjasto mahdollisti kuitenkin nopeasti toimivan prototyypin kehittämisen heti alkuun, ja siitä käytettyjen osien korvaaminen myöhemmin onnistui helposti.

2.2 Kehitystyökalut ja projektin hallinta

Projektin toteutuksessa käytettiin uusinta C++14-standardia ja hyödynnettiin C++-ohjelmointikielen tarjoamia moderneja ominaisuuksia, kuten automaattista palautustyypin määritystä ja lambdoja eli anonyymejä funktioita. Muistinhallinta toteutettiin käyttämällä smart pointereita. C++ valittiin kehityskieleksi siksi, että haluttiin tehokas ohjelmointikieli pelin suorituskykyä vaativien algoritmien ja useiden tekoälyn ohjaamien objektien logiikan päivityksen raskauden vuoksi.

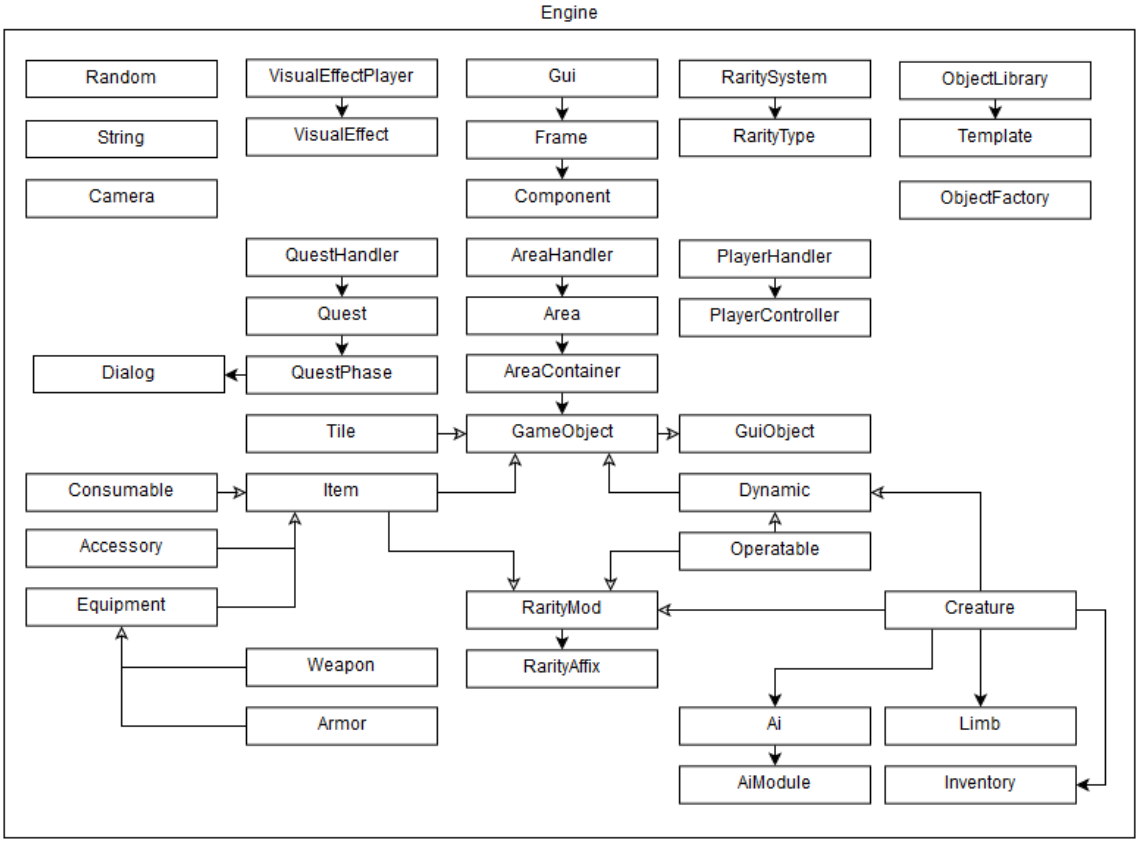
Pelin ohjelmointiin ja testaukseen käytettiin Visual Studion tarjoamia työkaluja. Versionhallintaan käytettiin git-versionhallintaa, ja lähdekoodi pidettiin GitHub-palvelussa.

Projektia toteutettiin siten, että haluttuja ominaisuuksia lisättiin listaan, josta niitä toteutettiin yksi kerrallaan toteutuksen kokonaisuuden kannalta järkeväksi todetussa

järjestyksessä. Ennen laajempien ominaisuuksien ohjelmoimista kirjattiin tarvittavien muutoksien vaikutus jo olemassa olevaan koodiin, jotta ominaisuuden toteuttamisen tarvitsemien muutosten tekeminen olisi helpompaa eikä muutosta tarvitsevia kohteita tarvinnut etsiä useaan kertaan. Tässä vaiheessa myös usein havaittiin arkkitehtuurillisesti parempia vaihtoehtoja ominaisuuden toteuttamiselle.

3 Arkkitehtuuri

Pelisimulaation ytimenä toimii pelimoottori, joka on määritelty Engine-luokassa. Kaikki pelin luokat pääsevät käsiksi moottoriin ja voivat hyödyntää siellä määriteltyjä globaaleja parametreja. Sen kautta luokat myös pääsevät käsiksi moottorin moduuleihin, kuten käyttöliittymään, objektikirjastoon, satunnaislukugeneraattoriin ja tekstin käsittelyn apuluokkaan. Moottorin luokat ja niiden väliset yhteydet esitetään kuvassa 1.



Kuva 1. Suuntaa antava kuvaus pelin arkkitehtuurista. Kaaviossa tummat nuolet kuvaavat luokkien välisiä navigointisuuntia ja vaaleat nuolet luokkien perintää.

Simulaatio mallinnettiin käyttäen perintäluokkiin jaettuja peliobjekteja. Peliobjekteja ovat kaikki pelin simulaatioon vaikuttavat objektit, eli maaston osat, tavarat, olennot ja operoitavat objektit. Simulaation manipulointi on toteutettu käyttöliittymän kehysten ja näppäimistön komentojen kautta. Objekteja simuloidaan pelin alueella, joka koostuu kaksiulotteisesta tasosta johon peliobjekteja lisätään, liikutetaan ja poistetaan. Alueet generoidaan, ja olentojen dialogit saadaan pelaajalla meneillään olevan tehtävän määritysten mukaan. Tekoäly jaettiin erillisiin moduuleihin, jotka käsittelevät tekoälyn ohjaaman objektin toimintaa tietyssä tilanteessa, kuten taistelussa tai rakennuksen sisällä olemista.

Pelin merkittävänä ominaisuutena on tavaroiden ja hahmojen harvinaisuus, mikä määrittää niiden saamien tilastojen suuruuden ja tekee harvinaisemmista tavaroista pelaajalle hyödyllisempiä ja olennoista haastavampia vastustajia. Harvinaisuus toteutettiin liitettävänä osana pelin objekteihin moniperinnän kautta. Pelissä eteneminen vaatii harvinaisten tavaroiden keräämistä, jotta pelaajan olennon on mahdollista pärjätä vastustajille, jotka ovat progressiivisesti haastavampia tehtävän myöhemmillä alueilla.

4 Toteutus

4.1 Pelimoottori

Pelimoottori pilkottiin useisiin avustaviin moduuleihin, jotka käsittelevät pelimoottorin erillisiä alueita. Näitä ovat muun muassa pelialueen hallinta, käyttöliittymä, satunnaislukugeneraattori, pelaajan toimintojen hallinta, peliobjektien kirjasto ja tehdas, harvinaisuusjärjestelmä, visuaalisten efektien hallinta sekä pelin kamera, jota käytetään määrittämään, minkä alueen pelaaja kulloinkin pelialueesta näkee. Engine-luokka sisältää myös pelissä käytetyt globaalit muuttujat. Niillä voidaan helposti säätää pelin simulaation parametreja, kuten tavaroiden alkuarvoa niitä kaupattaessa, olentojen kestävyyttä tai esimerkiksi kannettujen objektien painon aiheuttamaa rasitusta olentojen liikkumiseen.

Pelimoottorissa suoritettaan pelin pääsilmukkaa, jossa suoritetaan grafiikan piirto, partikkeliefektien simulointi ja vuorojen, tekoälyn sekä pelaajan näppäimistön kautta antamien komentojen käsittely. Grafiikan piirto tapahtuu lähettämällä render-kutsu

ensin alueelle ja alueelta jokaiselle kameran näkymässä olevalle objektille. Sitten kutsutaan visuaalisten efektien piirtoa ja lopuksi suoritetaan käyttöliittymän piirto.

Pelimoottori käsittelee myös vuorojen vaihtamisen pelaajan suorittaessa toimintoja, jotka vaikuttavat pelin simulaation ajan etenemiseen. Näitä toimintoja ovat käveleminen, hyökkääminen, ovien avaaminen ja sulkeminen sekä tavaroiden poiminnan aloittaminen. Simulaatio päivitetään pelaajan toiminnosta seuraavaan vuoroon, jonka jälkeen kukin tekoälyn ohjaama olento suorittaa oman vuoronsa tekoälyn logiikan valitsemalla tavalla, ja tämän jälkeen pelimoottori palaa jälleen odottamaan pelaajan seuraavaa komentoa. Pelin simulaatio saadaan alustettua kutsumalla pelimoottorin newGame-metodia, jolloin tallennetut tiedot tyhjennetään, pelaajan olento generoidaan ja valitun tehtävän mukaan generoidaan uusi alue jonne pelaajan olento asetetaan.

4.2 Grafiikka

Grafiikan piirto toteutettiin hyödyntämällä libtcod-kirjaston tarjoamaa konsolia. Konsoli on jaettu määritellyn koon mukaisiin ruutuihin, jotka täyttävät koko konsolin pituuden ja leveyden. Ruutuihin voidaan määritellä sen sisältämä merkki, merkin väri, sekä ruudun taustan väri. Lisäksi värin ja merkin lisäyksen yhteydessä voidaan antaa erillisiä lipukkeita, joilla voidaan määritellä tapa, jolla väri sekoitetaan jo ruudulle asetettuun väriin.

Simulaation kamera tarjoaa näkymän simulaation alueeseen. Kamera seuraa pelaajan ohjaamaa olentoa ja konsolin ruudut päivitetään kuvastamaan kameran näyttämiä koordinaatteja sen sillä hetkellä näyttämästä alueen osasta. Käyttöliittymän kehykset piirretään konsoliin alueen ruutujen päälle.

Alueen ruutuja piirrettäessä niille asetetaan valon intensiteetti, jonka avulla pelaajalle esitetään sen hallitseman olennon näkökentän laajuus. Valon intensiteetti visualisoidaan piirtämällä konsoliin musta ruutu kaikkien alueen ruutujen päälle ja asettamalla mustaa ruutua piirrettäessä sille alfa-kerroinlipuke, jossa intensiteetti kuvastaa sen läpinäkyvyyttä. Mustan ruudun läpi suodatetaan tällöin alla olevan ruudun väri alfa-kertoimen mukaan.

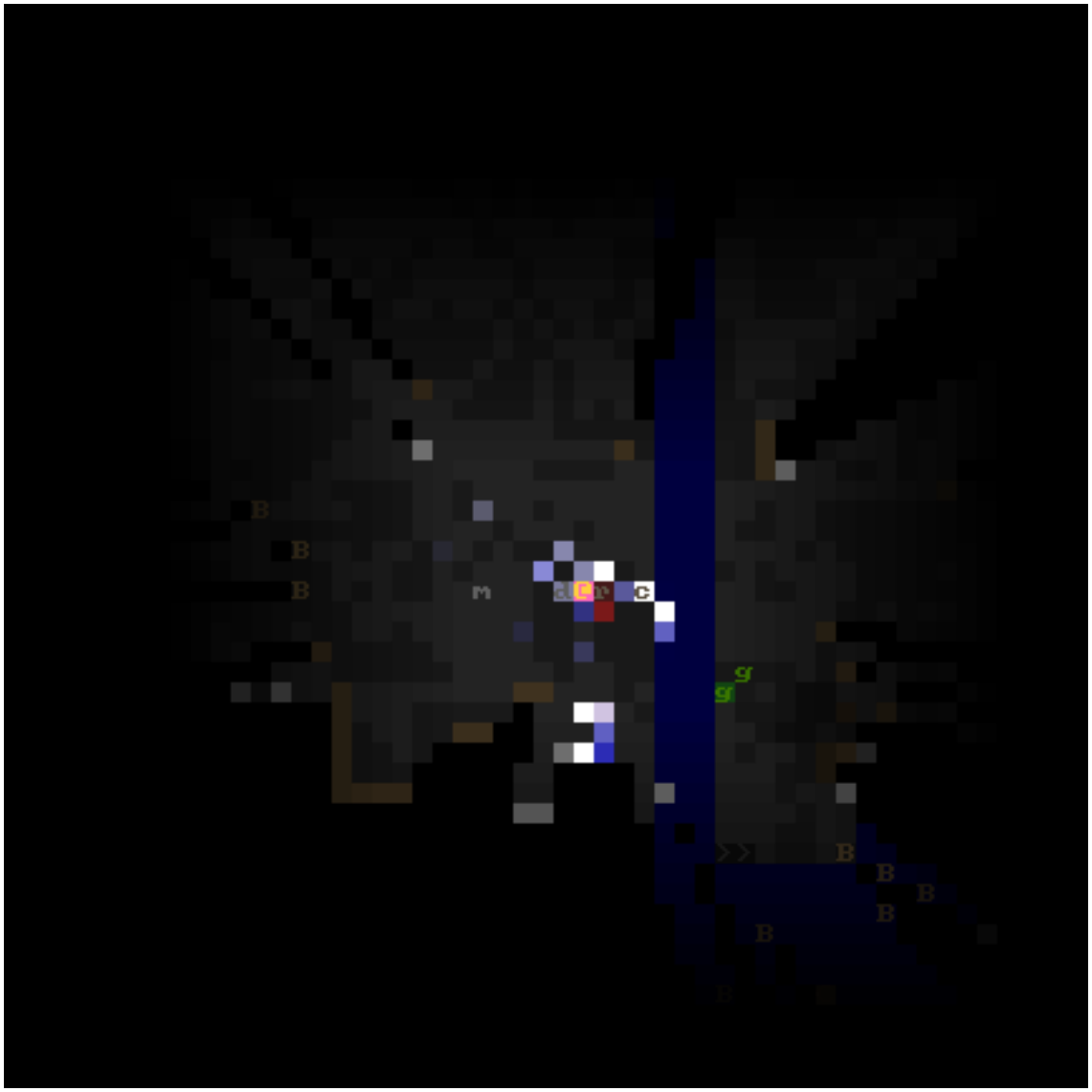
Grafiikkaa ja partikkeliefektejä päivitetään jatkuvasti tasaisella nopeudella, vaikka simulaation toiminnallinen logiikka toimii vuoropohjaisesti. Tällöin pelaajalle voidaan esittää enemmän informaatiota vuoroista erillään olevien efektien muodossa ja näyttää alueella päällekkäin olevat tavarat piirtämällä niitä vuorotellen alueen kohtaan, jossa tavarat sijaitsevat.

Partikkeliefektit

Partikkelit on kuvattu konsolissa sen koordinaateissa sijaitsevan ruudun taustaväriä vaihtamalla. Partikkeliefektit määriteltiin VisualEffect-luokassa, jota laajentamalla luotiin erilaisia efektejä. Luokissa määriteltiin efektin partikkelien maksimi-ikä, väri, määrä ja niiden suunta sekä liikkumis- ja häivytysnopeus. Partikkeleja luotaessa pelimoottori tarkistaa, että partikkeli näkyy pelaajan näkökentässä, muutoin partikkelien luominen jätetään pois, koska niillä ei ole peliin toiminnallista vaikutusta.

Jokaisella päivityksellä efektin partikkeleiden ikää kasvatetaan ja lisätyn iän mukaan lasketaan partikkelin seuraava positio pelialueella sen määrätyn liikkumisnopeuden ja satunnaisluvun mukaan. Myös partikkelin väriä ja häivytystä voidaan muuttaa, ja lopulta määritellyn maksimi-iän tullessa täyteen partikkeli poistetaan. Kun kaikki partikkelit on poistettu efektistä, poistetaan myös itse efekti VisualEffectPlayer-luokasta, jolloin se ei ole enää grafiikan päivityksessä mukana ja sen varaamat resurssit vapautuvat.

Partikkeleilla saadaan aikaan näyttäviä efektejä kaksiulotteisessakin pelissä, ja ne lisäävät pelaajalle informaatiota pelin tapahtumista. Partikkeleilla kuvastetaan pääasiassa pelin hyökkäystoimintojen iskuja, olentoihin vaikuttavia taikojen tai kulutettavien tavaroiden kautta hankittuja efektejä. Efektien avulla on helpompi päätellä, mitä kulloinkin tapahtuu, vaikka grafiikka onkin vain merkeillä ja väreillä mallinnettua. Kuvassa 2 havainnollistetaan partikkeliefektien ilmenevyys.



Kuva 2. Taistelussa syntyneitä partikkeliefektejä

4.3 Käyttöliittymä

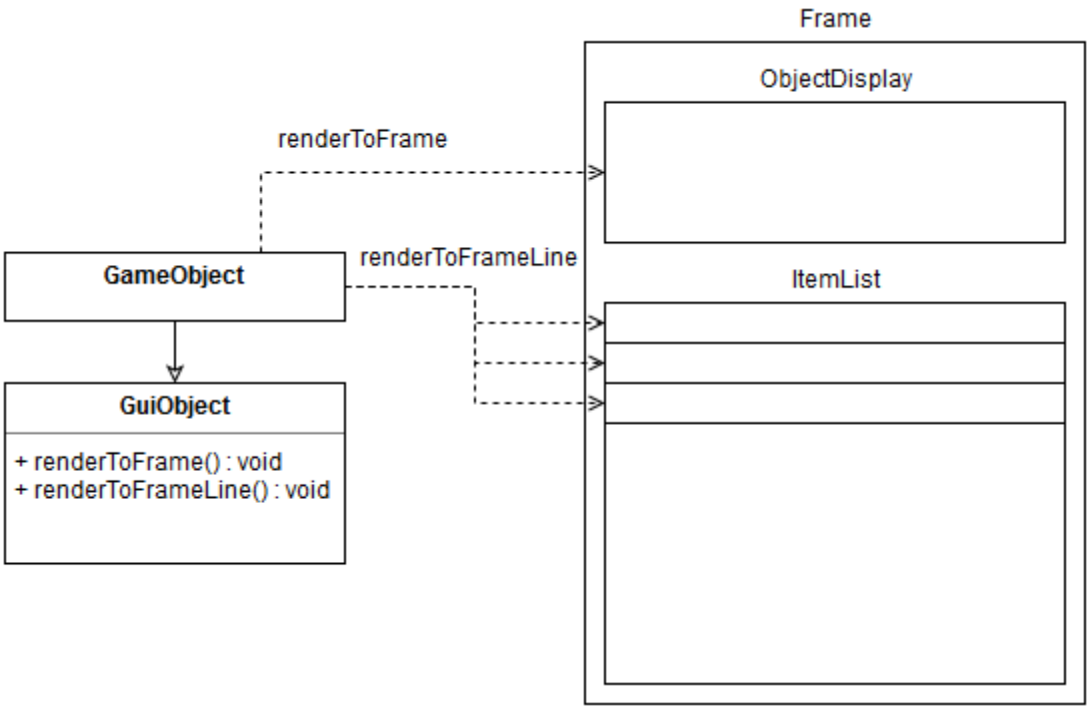
Ohjaus

Pelin ohjaus toteutettiin toimivaksi pelkästään näppäimistöllä. Pelaajan olennon liikkumisen suunnan valinta ja valikoiden sekä dialogien kursorin liikuttaminen toimii numpad-näppäinten numeroilla. Numerot yhden ja seitsemän välillä vastaavat olennon liikuttamissuuntaa, ja kehyksissä valintoja selataan numeroilla 4 ja 6, listan selaus numeroilla 8 ja 2 ja valinnan hyväksyntä numerosta 5. Tämä tekee ohjauksen oppimisen jälkeen pelaamisesta nopeatempoista. Sellaiset toiminnot kuin esimerkiksi

hyökkäyksen aloittaminen, pelaajan kantamien tavaroiden tarkastelu tai oven avaaminen on liitetty toimintoa kuvaavaan kirjaimeen. Monet komennoista avaavat pelin näkymään kehyksen, jonka kautta toiminto suoritetaan. Näkymässä esitetyt kehykset saa myös pois näkyvistä painamalla kehyksen avaamiseen liitettyä näppäintä uudelleen. Näppäinten komentojen asettamiseen toteutettiin KeyMapping-luokka, josta on mahdollista helposti vaihtaa toimintojen näppäimiä.

Käyttöliittymäobjekti

Kaikki pelin objektit, jotka haluttiin esittää käyttöliittymässä, on moniperinnän kautta laajennettu käyttöliittymäobjektista, joka on määritelty GuiObject-luokassa. Laajennettaessa objekti toteuttaa renderToFrame-metodin, jolla se voidaan piirtää neliön muotoiseen ruutuun, ja renderToFrameLine-metodin, jolla se voidaan piirtää yhden rivin korkuiselle viivalle. Laajemmassa ruudussa voidaan esittää objektista tarkempia tietoja ja viivalle piirrettäessä lyhyt tiivistelmä objektin tärkeimmistä tilastollisista ominaisuuksista. Käyttöliittymäobjektin esittäminen kehyksessä havainnollistetaan kuvassa 3.



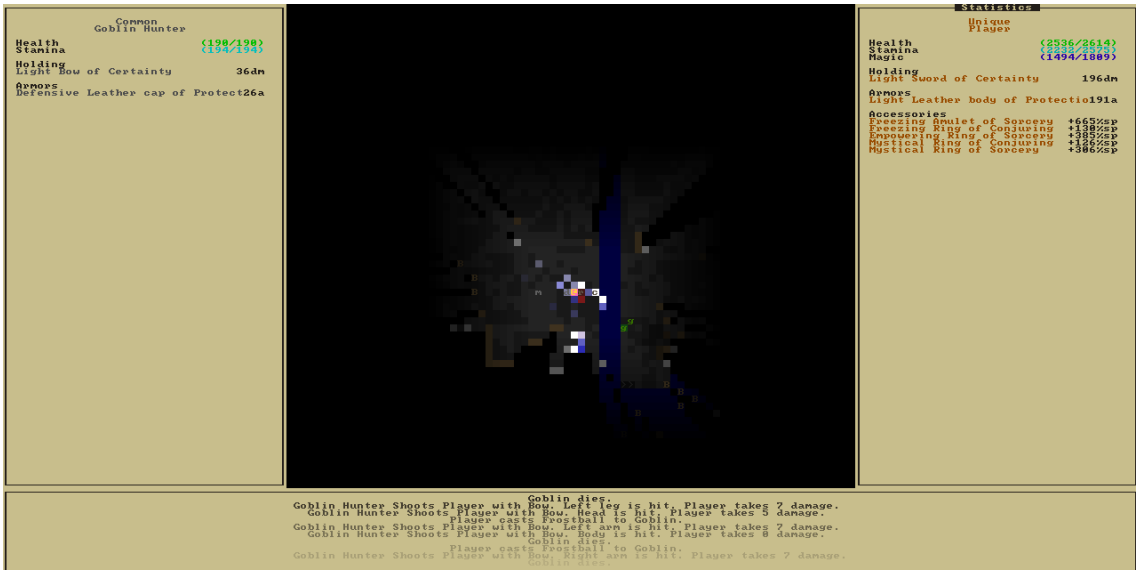
Kuva 3. Kuvaus peliobjektin esittämisestä käyttöliittymän kehyksessä. Vaalea nuoli kuvaa luokan perintää. Katkoviivalla piirretyt nuolet kuvaavat metodin käyttöä.

Kehykset

Pelin käyttöliittymä koostuu kehyksistä, joissa pelaajalle esitetään tietoa pelin objekteista, ja niiden kautta tapahtuu osa pelin toiminnoista, kuten esimerkiksi tavaroiden poiminta, kaupankäynti, dialogit muiden olentojen kanssa ja taistelutaitojen valinta. Kehyksillä myös annetaan pelaajalle lisätietoja simulaation tapahtumista ja objekteista. Kehykset laajennettiin GuiFrame-luokasta. Kehysten käyttöä pelin käyttöliittymässä havainnollistetaan kuvassa 4.

Kehykset koostettiin komponenteista. Lista-komponenteilla esitetään lista tavaroista, taidoista tai toiminnoista. Listat pitävät muistissaan valitun indeksin ja tukevat vieritystä, jolloin kuvaruutua suurempiakin listoja pystytään esittämään pelaajalle. Listojen yhteydessä esitetään valittavan kohteen kuvaus ja lyhennetyt tilastolliset tiedot hyödyntäen käyttöliittymäobjektin renderToFrameLine-metodia sekä valitun indeksin kohdalla operaatiot, joita kohteeseen voi käyttää. Valittu operaatio esitetään objektin kuvauksen kanssa samalla viivalla, ja valittua operaatiota voi vaihtaa selaamalla mahdollisia operaatioita horisontaalisti selausnäppäimillä.

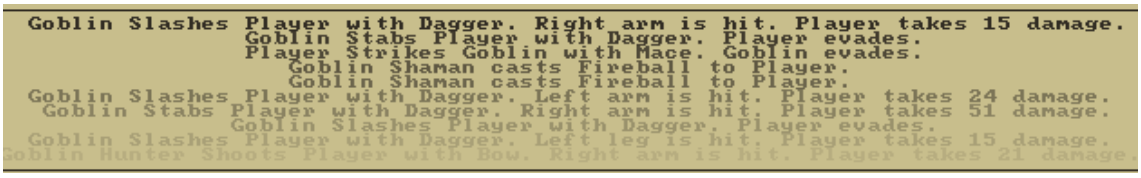
Erillisessä objektin esittämiskomponentissa, joka on määritelty GuiGameObjectDisplay-luokassa esitetään pelin objektista lisätietoja, kuten sen nimi, harvinaisuus ja tilastolliset ominaisuudet tarkemmin. Komponentissa hyödynnetään käyttöliittymäobjektin renderToFrame-metodia. Kehyksiin asetelluille komponenteille on annettu kehykseen suhteelliset kohdat, joissa komponentit esitetään, joten kehykseen liitetyt komponentit siirtyvät kehyksen mukana, jos sen paikkaa pelin näkymässä vaihdetaan. Kehysten kautta pelaajan antamat komennot ohjataan oikeisiin komponentteihin, jolloin esimerkiksi kahden listan samanaikainen esittäminen ja niiden välillä liikkuminen onnistuu.



Kuva 4. Pelin yleisnäkymä

Loki

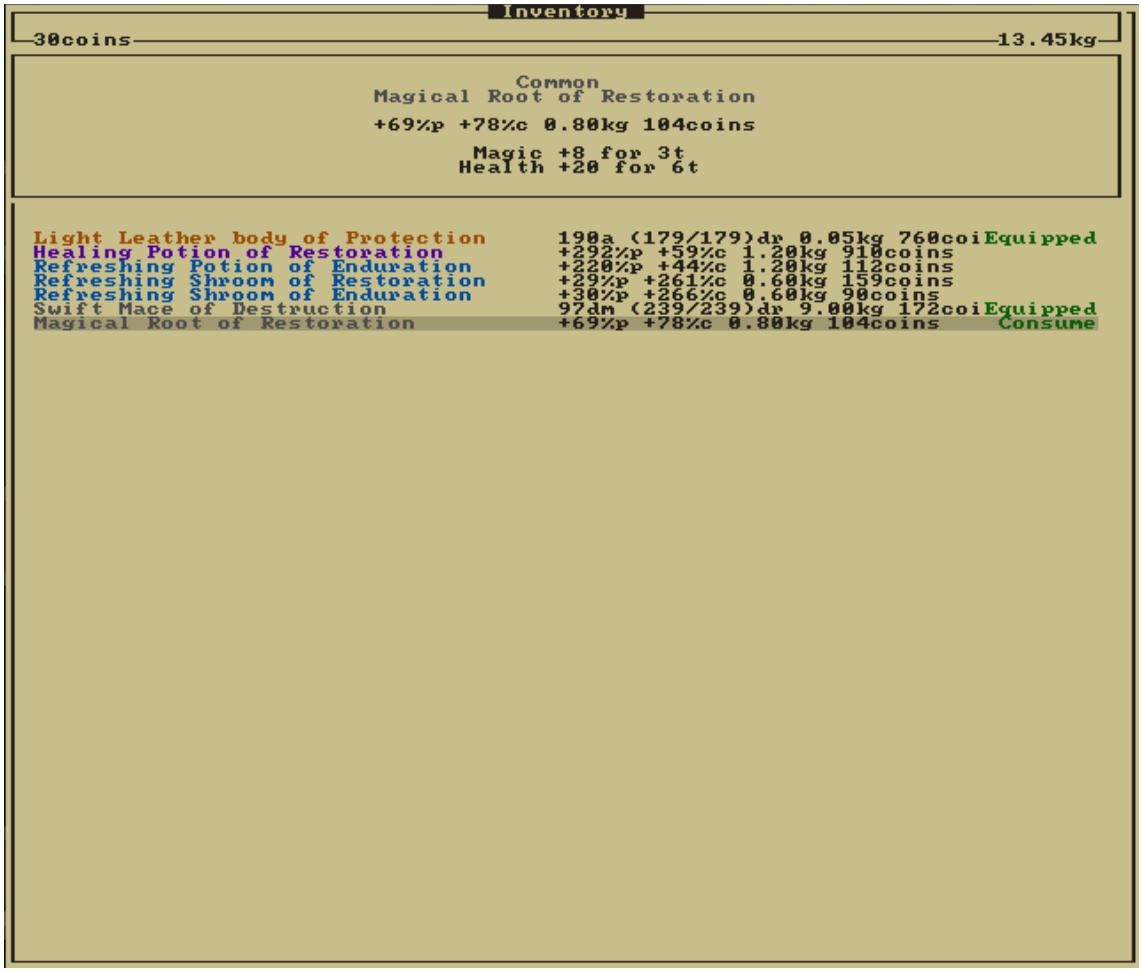
Simulaatio tulostaa lokiin tulosteita, jotka kuvaavat simulaation tapahtumia. Tulosteet kuvaavat olentojen hyökkäyksiä, dynaamisten objektien tuhoutumista, ilmoituksia pelaajan olennon uupumisesta, tavaran tuhoutumisesta ja niin edelleen. Uuden tulosteen seurauksena vanhat tulosteet siirtyvät dialogissa alaspäin ja häivyttyvät lokin taustaan. Tämä parantaa uusien tulosteiden luettavuutta. Lokin saa myös pois näkyvistä, jos pelaaja kokee sen häiritseväksi tai näkymää peittäväksi. Kuvassa 5 havainnollistetaan lokin tulosteet.



Kuva 5. Log-kehyksen näkymä taistelun aikana.

Inventaario

Inventaario-kehyksen kautta pelaaja voi tarkastella, pukea, riisua, pudottaa ja käyttää ohjaamansa olennon tavaroita. Inventaariossa esitetään myös listasta valitun tavaran tarkemmat tiedot. Valittavat operaatiot riippuvat tavaran tyypistä. Inventaario-kehys havainnollistetaan kuvassa 6.



Kuva 6. Pelaajan inventaario-kehys

Tavaroiden poiminta

Pelaaja voi poimia alueelta tavaroita ohjaamalleen olennolle tavaroiden poiminnan kehyksen kautta. Kehyksessä näytetään lista tavaroista, jotka sijaitsevat alueella olennon kohdassa. Listasta pelaaja voi poimia haluamansa tavarat ja jättää muut paikoilleen.

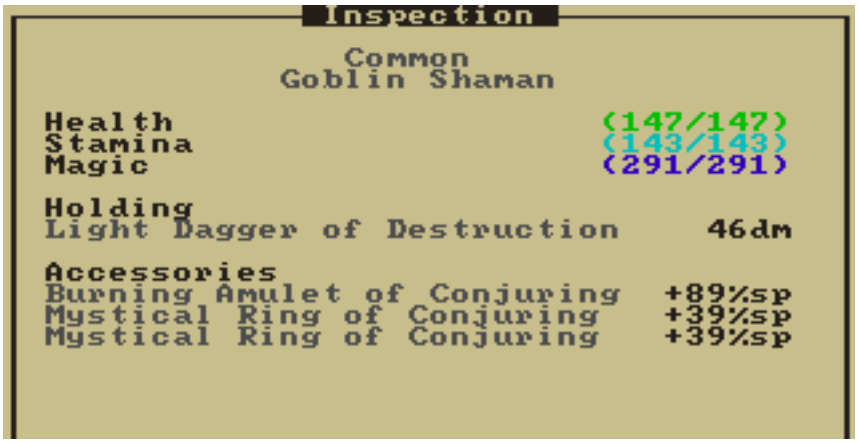
Tarkastelu

Inspection-luokassa määritelty kehys koostuu objektin esittämiskomponentista ja kursorista, joka esitetään pelin alueen simulaation päällä. Tällä kehyksellä pelaaja voi

tarkastella simulaation alueella olevia objekteja, jotka ovat pelaajan näkökentässä. Kehys havainnollistetaan kuvassa 7.

Kehyksen komponentissa esitetään kursorin kohdalla olevan objektin tarkemmat tiedot. Esimerkiksi jos objekti on olento, näytetään sen tilastot, pukemat tavarat ja efektit, jotka siihen vaikuttavat. Kehyksessä on tarkistus sille, että vain pelaajan olennon näkökentässä olevia objekteja voi tarkastella. Jos näkökentän ulkopuolella olevaa aluetta yrittää tarkastella, piilotetaan objektin tietoja esittävä komponentti näkymästä.

Objektien tarkastelu aukeaa myös automaattisesti hyökkäystoimintoa suoritettaessa, jolloin avatussa kehyksessä näytetään hyökkäyksen kohteen tarkemmat tiedot. Tämä auttaa pelaajaa valitsemaan kohteen, jota vastaan on sopivinta hyökätä ja nopeuttaa pelaamista, koska tarkastelun kehystä ei erikseen tarvitse avata. Oletuksena pelin näkymässä esitetään jatkuvasti kehys, jossa esitetään pelaajan ohjaaman olennon tiedot.



Kuva 7. Olennon tarkastelu Inspection-kehyksessä

Hyökkäystoiminnon suorittaminen

Hyökkäystä aloitettaessa pelaaja valitsee ohjaamaltaan olennolta taidon, jota halutaan käyttää. Taito valitaan kehyksestä, johon sitä avattaessa olennon taidot haetaan listaan dynaamisesti sen pukemien tavaroiden mukaan. Taidon valinnan jälkeen avataan uusi kehys, jossa valitaan taitoon kuuluva hyökkäystoiminto. Valittu taito ja hyökkäystoiminto jäävät kehyksen muistiin. Myöhemmin pelaajan toistaessa hyökkäystoimintoa käytetään kehyksen muistissa olevan taidon hyökkäystoimintoa, jos

sitä on vielä mahdollista käyttää, eli olennolla on esimerkiksi edelleen puettuna ase, johon valittu taito ja hyökkäystoiminto kuuluvat.

Hyökkäystä aloitettaessa kursori kohdistetaan lähimpään vastustajaan. Hyökkäyksen kohdetta on mahdollista vaihtaa käyttämällä selausnäppäimiä, jolloin pelaajan ei itse tarvitse kuljettaa kursoria hyökkäyksen kohteen päälle. Kohde, jota vastaan pelaaja hyökkää, pidetään kehyksen muistissa ja pidetään ensimmäisenä kohteena seuraavaa hyökkäystoimintoa suoritettaessa. Tämä estää pelaajan priorisoidun kohteen tahattoman vaihtumisen, jos esimerkiksi toinen mahdollinen kohde kulkee alkuperäistä kohdetta lähemmäksi suhteessa pelaajaan.

Dialogit

Dialogi-kehysten kautta hallitaan pelaajan ja olennon välinen kommunikointi. Dialogia avattaessa tarkastetaan, onko pelaajan ohjaaman olennon viereisellä maaston osalla olentoja, joiden kanssa aloittaa dialogi. Jos olentoja on vain yksi, dialogi aloitetaan sen kanssa. Jos vierekkäisiä olentoja on useampia, näytetään pelaajalle kursori, jolla dialogin kohde valitaan hyökkäyksen kohteen tapaan selaamalla mahdollisia kohteita.

Dialogin kautta esitetään keskusteluvaihtoehdot, joita pelaaja voi olentotyypin kanssa valita. Vaihtoehdot on määritelty simulaation käynnissä olevan tehtävän mukaan. Tehtävän määrityksissä olentojen malleille, joiden kanssa dialogin voi aloittaa, on asetettu vaihtoehdot, joita tehtävän aikane olennon mallin kanssa voi käydä. Kaikki olennot, jotka on generoitu mallin perusteella, avaavat samat keskusteluvaihtoehdot. Dialogin keskusteluvaihtoehtojen kautta pelaaja voi esimerkiksi valita kaupankäynnin aloittamisen tai jatkaa dialogia eli valita keskusteluun liittyvän operaation tai lopettaa dialogin.

Kauppa

Kauppa-kehykseen on aseteltu kaksi tavaralistaa ja alueet joissa esitetään kauppaa käyvien olentojen rahat, sekä pelaajan kantokapasiteetti. Listoihin asetetaan kauppaa käyvien olentojen tavarat, ja pelaaja voi valita ostettavat ja myytävät tavarat. Tavaroiden arvo on laskettu dynaamisesti niiden arvioidun hyödyllisyyden mukaan, ja valittujen tavaroiden osto, ja myynti, arvojen erotus esitetään pelaajalle kehyksessä. Kauppatapahtumaa hyväksyttäessä arvojen erotus lasketaan osapuolten rahoihin ja

valitut tavarat siirretään omistajilleen. Ennen kauppatapahtuman hyväksymistä tarkistetaan, että molemmat olennot jaksavat kantaa uudet tavaransa ja että ostajan rahat riittävät valittuihin tavaroihin. Kauppa-kehys havainnollistetaan kuvassa 8.

Kuva 8. Kauppa-kehyksen näkymä

4.4 Tehtävät

Pelaajalla on kulloinkin aktiivisena yksi tehtävä, jonka määritysten mukaan pelialuetta generoidaan pelaajan edetessä alueelta toiselle.

Tehtävät laajennettiin Quest-luokasta, ja ne koostuvat useista eri vaiheista, jotka laajennettiin QuestPhase-luokasta. Jokaisessa vaiheessa on määritelty kyseisen vaiheen alue ja sen yleinen koostumus, eli onko kyseessä esimerkiksi metsä tai luolasto, sekä alueella esiintyvät objektit, kuten olennot ja arkut, ja näiden esiintymistodennäköisyydet ja harvinaisuus.

Pelaajan edetessä tehtävän alueelta toiselle ladataan käynnissä olevan tehtävän seuraava vaihe ja siinä määritelty alue ja sen objektit generoidaan simulaation alueelle ja pelaajan olento siirretään alueen keskipisteeseen.

Tehtävän vaiheiden alueet voidaan asettaa persistoiduiksi, eli tehtävän vaiheen alueelle palattaessa ladataan sen edellinen tila muistista. Tätä on käytetty pitämään peliin luodun tehtävän kylä tallennettuna, jotta esimerkiksi pelaajan sinne jättämät

tavarat eivät katoaisi ja kaupoista löytyvät tavarat ovat edelleen samat pelaajan niihin palatessa.

4.5 Pelaaja

Pelaaja vaikuttaa pelisimulaatioon hallitsemalla olentoa PlayerController-luokassa määritellyllä moduulilla, joka käsittelee pelaajan komennot näppäimistöltä ja kääntää ne olennon toiminnoiksi. Pelaajalla on käytössään toimintoja, joita tekoälyn ohjaamilla olennoilla ei ole. Näitä toimintoja ovat tavaroiden poiminta, dialogin aloittaminen ja alueelta toiselle siirtyminen. Pelaaja ohjaama olento mallinnetaan simulaatiossa muuten samalla tavalla kuin tekoälynkin ohjaamat olennot.

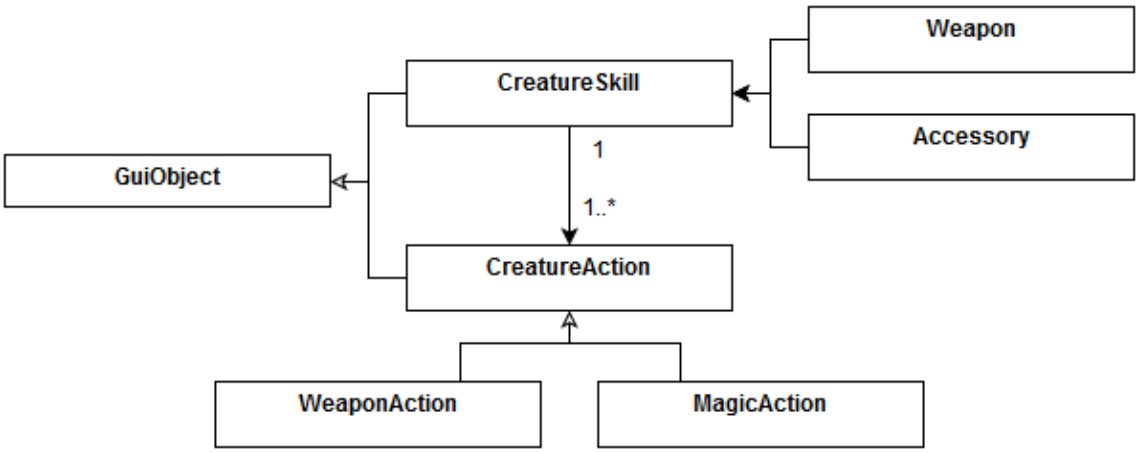
4.6 Tilastot

Erilaisilla tilastoilla kuvataan peliobjektien tilaa. Tilastoilla on mitattu objektien kestävyyttä, jaksamista, kantokapasiteettia, taikavoimaa, niiden aiheuttamaa vahinkoa, puolustuskykyä, painoa ja arvoa. Tavaroiden tilastot lasketaan dynaamisesti niihin liitettyjen harvinaisuus-affiksien mukaan, ja olentojen tilastot lasketaan dynaamisesti niiden pukemien tavaroiden vaikutusten mukaan.

Kun tilastoja generoidaan pelin objekteille, käytetään niiden asettamisessa prosentuaalisia osuuksia maksimiarvoista, jotka on määritelty pelin moottorin globaaleissa muuttujissa. Näin tilastot voidaan helpommin pitää tiettyjen suhteellisten rajojen sisälle, ja pelin vaikeutta tasapainottaessa arvoja on helpompi muuttaa.

4.7 Toiminnot

Toiminnot ovat olentojen kutsumia tapahtumia, joilla voidaan aiheuttaa toiseen dynaamiseen objektiin vahinkoa, tai esimerkiksi taikoja, jotka antavat kutsujalleen parantavia efektejä. Toiminnot laajennettiin CreatureAction-luokasta. Toiminnot muokkaavat dynaamisten objektien tilastoja, tulostavat toiminnon suorituksen ja seurauksen tiedot lokiin ja toistavat visuaalisen efektin. Toiminnot on liitetty erilaisiin taitoihin joita olennot käyttävät. Toimintojen liittyminen peliobjektiin havainnollistetaan kuvassa 9.



Kuva 9. Kuvaus tavaroihin liitetystä taidosta ja sen sisältämistä toiminnoista. Vaaleat nuolet kuvaavat luokan perintää. Tummat nuolet kuvaavat luokkien välisiä navigointisuuntia.

Toiminnossa on määritelty sen nimi, lokiin tulostettava tuloste, tilastot, joita suorittamiseen vaaditaan, olennon taidon vaikutus toimintoon, suoritusetäisyys, visuaalinen efekti ja kohteen tyyppi, jolla voidaan valita, voiko esimerkiksi taialla valita kohteeksi sen suorittajan, jolloin esimerkiksi olento voi valita parantavien taikojen kohteeksi itsensä. Toimintoa kutsuttaessa sille annetaan parametreina sitä kutsuva olento, olennon taidon vaikutus suoritukseen, objekti, jota suoritukseen käytetään, kuten ase, ja kohde, johon, toiminto kohdistetaan. Suorituksen yhteydessä tarkistetaan, onko kohde riittävällä etäisyydellä ja riittävätkö kutsuvan olennon tilastojen suureet toiminnon suorittamiseen. Lopuksi jos suoritus onnistuu toistetaan visuaalinen efekti.

Ase-toiminnot

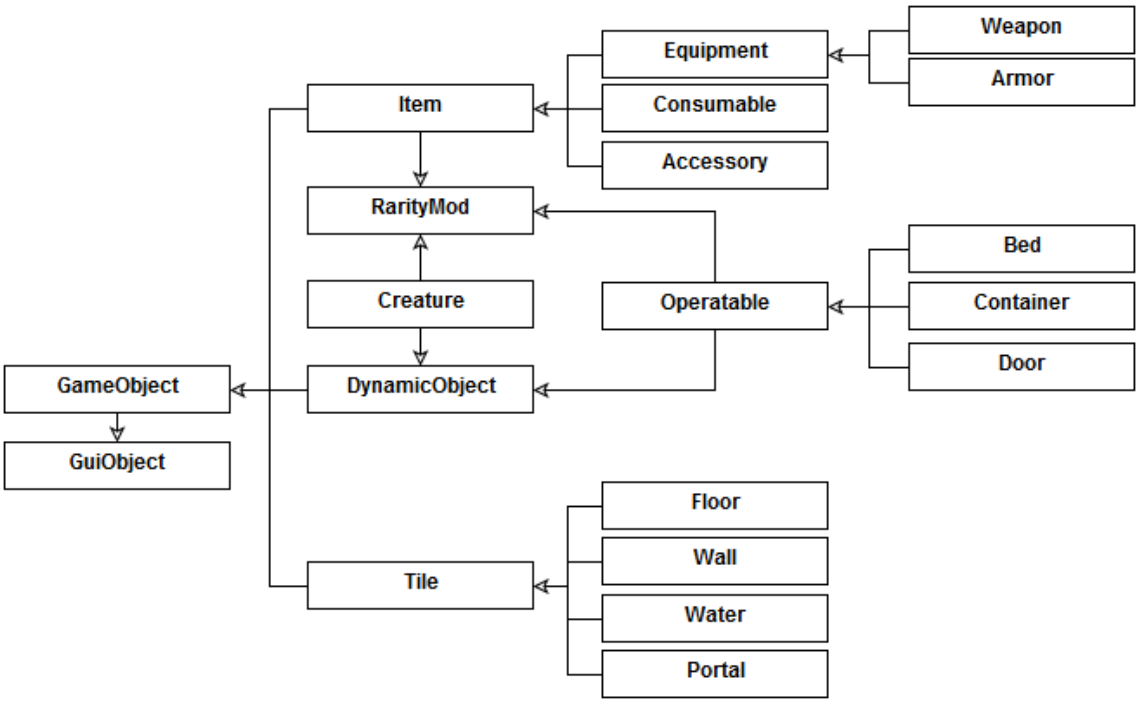
Ase-toiminnot laajennettiin toiminnosta. Ase-toiminnot, joita olento voi suorittaa, tulevat sen käyttöön valituista aseista. Niiden suoritukseen vaikuttavat kutsuvan olennon taito, jaksaminen, aseen kunto ja tarkkuus, johon vaikuttaa olennon jaksaminen, valittu ase ja satunnaisuus. Jos olennon jaksaminen riittää toiminnon suorittamiseen, vahingoitetaan valittua kohdetta aseen aiheuttaman vahingon verran ottaen huomioon suojaavat varustukset. Lopuksi hyökkäykseen käytettyyn aseeseen aiheutetaan kulumaa määritellyn verran. Jos ase tuhoutuu kuluman seurauksena, tulostetaan tieto siitä lokiin. Olennon suorittaessa aseella hyökkäystä valitaan aseeseen liitetty satunnainen ase-toiminto. Toiminnot kuvaavat erilaisia iskuja, joita aseella hyökätessä voi tapahtua. Toiminnot asetettiin asetyypin määrityksiin objektikirjastossa.

Taika-toiminnot

Taika-toiminnot laajennettiin toiminnosta. Taika-toiminnot, joita olento voi suorittaa, ja niiden vaikutusten suuruus tulevat olennon pukemista koristavista tavaroista. Niissä on toiminnon määritysten lisäksi, määritelty efektit, joita ne kohteelle aiheuttavat. Niiden suoritukseen vaikuttavat kutsuvan olennon taito, taikavoiman määrä ja jaksaminen. Jos kutsujan taikavoima ja jaksaminen riittävät taidon suoritukseen lisätään, kohteelle määritellyt efektit, joiden vaikutuksen suuruuteen ja kestoon vaikuttaa kutsujan taito. Taika-toiminnot liitettiin koristaviin tavaroihin harvinaisuus-affikseilla. Mahdolliset toiminnot määriteltiin objektikirjastossa, josta ne liitettiin harvinaisuus-affikseihin.

4.8 Peliobjektit

Kaikki peliobjektit periytyvät GameObject-luokasta, jossa on määritelty, mitä tyyppiä objekti on, kuten onko objekti ovi, seinä, olento, sänky tai arkku. Objektin tyyppiä käytetään selvittämään, miten simulaation kuuluisi sitä käsitellä. Peliobjektit periytyvät moniperinnän kautta myös käyttöliittymäobjektista ja objektin tyypistä riippuen harvinaisuus-luokasta. Objektissa käsitellään sen piirto simulaation näkymään, ja jokainen käyttöliittymässä esitettävä objekti määrittelee oman piirtonsa käyttöliittymään toteuttamalla käyttöliittymäobjektin metodit. Peliobjektien perintä havainnollistetaan kuvassa 10.



Kuva 10. Kuvaus peliobjektien perinnästä. Nuolet kuvaavat luokan perintää.

Maaston osa

Pelialueet koostuvat maaston osista, jotka on laajennettu Tile-luokasta. Luokassa määritellään, onko osa olennon kuljettavissa, kuinka raskasta sen läpi kulkeminen on ja minkä efektin siitä kulkeminen aiheuttaa. Esimerkiksi vedessä kulkeminen on olennoille raskasta ja aiheuttaa efektin, joka kuvaa veden loisketta. Poikkeuksellinen maaston osa on portaali, jonka päällä ollessa pelaaja voi vaihtaa pelialuetta.

Tavara

Tavarat ovat objekteja, joita olennot voivat pukea, käyttää tai kuluttaa. Kaikki pelin tavarat, kuten varusteet, parantavat juomat ja amuletit laajennettiin Item-luokassa. Tavaroille on määritelty niiden pitämiseen olennolta vaadittu vapaiden raajojen määrä. Esimerkiksi raskaan miekan tai jousen pitämiseen vaaditaan kaksi kättä. Luokassa määritellään myös tavaran paino ja dynaamisesti sen arvo. Harvinaisuus on laajennettu luokkaan moniperinnän kautta, ja se vaikuttaa tavaran ominaisuuksiin sen tyypin mukaan.

Tavaroiden arvo kauppaa käydessä lasketaan tavaran arvioidun hyödyllisyyden mukaan. Hyödyllisyys on arvioitu lähes kaikkien tavaran tilastojen ja harvinaisuuden

mukaan. Esimerkiksi hyvin harvinainen ja paljon vahinkoa tuottava sekä kevyt miekka on rahallisesti arvokas, kun taas vain vähän parantava ja suhteellisen painava parannusjuoma ei ole niin arvokas kauppaa käydessä.

Puettavat tavarat

Puettaville tavaroille määriteltiin kestävyys. Puettavat tavarat laajennettiin edelleen suojavarusteisiin Armor-luokkassa, jossa tavaraan on lisätty määritys sen suojaavasta vaikutuksesta, jonka suuruuteen tavaran kestävyys vaikuttaa, ja aseisiin Weapon-luokassa, jossa on määritelty, mitä taitoa aseen käyttäminen olennolta vaatii, sekä sen aiheuttama vahinko, johon sen kestävyys vaikuttaa, ja aseen tarkkuus ja vaikutus käyttäjän jaksamiseen sitä käytettäessä.

Kulutettavat tavarat

Kulutettavissa tavaroissa on määritelty tieto kulutettavan tavaran vaikutuksen voimasta ja konsentraatiosta, jotka muokkaavat tavaraan liitettyjen harvinaisuus-affiksien vaikutusta tavaraa nautittaessa. Harvinaisuus kulutettavassa tavarassa nostaa näitä arvoja ja mahdollistaa parempien affiksien generoitumisen kulutettavaan tavaraan. Korkea arvo voimassa lisää affiksien tuomien efektien vaikutuksen suuruutta, ja korkea arvo konsentraatiossa lisää efektien vaikutuksen kestoa.

Koristavat tavarat

Koristavat tavarat vaikuttavat puettaessa olennon taikakykyihin. Luokassa on määritelty tavaran taikavoima, jolla se vaikuttaa tavaraan liitettyihin harvinaisuusvaikuttajiin. Koristavien tavaroiden avulla olento voi saada käyttöönsä harvinaisuus-affikseja, jotka mahdollistavat taikuuden käyttämisen. Harvinaisuus koristavissa tavaroissa voi luoda myös käytettävissä olevan taikakyvyn tehokkuuteen vaikuttavia harvinaisuus-affikseja. Useiden samoihin taikuuksin vaikuttaviin tavaroiden pukeminen vahvistaa taikuuden vaikutusta kumulatiivisesti.

Dynaaminen objekti

Dynaamisia objekteja ovat objektit, joihin olennot voivat toiminnoillaan vaikuttaa. Dynaaminen objekti on määritelty DynamicObject-luokassa. Luokassa on määritelty

objektin kestävyys, toistettava efekti sitä vahingoitettaessa ja viesti, jonka se tuhoutuessaan tulostaa käyttöliittymän lokiin, sekä sen määritys voiko olento tai valo kulkea sen läpi. Dynaamiselle objektille on myös asetettu harvinaisuus, joka vaikuttaa objektin tilastoihin ja sen omistamien tavaroiden harvinaisuuteen.

Operoitava objekti

Operatable-luokassa on määritelty operoitava objekti, joka on laajennettu dynaamisesta objektista ja siihen on lisätty määritys, mitä objekti tekee olennon sitä operoidessa.

Operoitavia objekteja ovat objektit, joita olennot voivat operoida erillisellä operointitoiminnolla omalla vuorollaan. Esimerkiksi ovet, joita olennot voivat avata ja sulkea, ja sängyt, joihin olennot voivat mennä nukkumaan, ovat operoitavia objekteja. Pelaajan avattavissa olevat arkut ja tekoälyn ohjaamien kauppiaiden käyttämät pelialueen koristeena toimivat objektit, kuten alasimet, uunit ja alkemistin pöydät, ovat operoitavia objekteja.

Olento

Creature-luokka laajennettiin DynamicObject-luokasta, ja siinä määriteltiin pelin olento. Olennot ovat tekoälyn tai pelaajan ohjaamia objekteja, jotka voivat vaikuttaa simulaation dynaamisiin objekteihin.

Olennolle on dynaamisen objektin ominaisuuksien lisäksi määritelty sen kestävyys, taikavoima ja sen pudottamien tavaroiden määrään vaikuttava muuttuja ja tyyppi, jonka mukaan olento voidaan tunnistaa esimerkiksi dialogia haettaessa olennolle. Olennoille on myös asetettu tavaroiden varasto, ohjaava tekoäly sekä raajat, joihin olennolle voidaan pukea tavaroita käytettäväksi. Olennon tilastot lasketaan dynaamisesti sen pukemista tavaroista. Tavaroista saadaan myös toiminnot, joita olento voi käyttää.

Olennon tuhoutuessa sen harvinaisuus vaikuttaa pudotettujen tavaroiden harvinaisuuteen, ja tavaroiden määrä muodostuu globaalin muuttujan ja olennon oman määritellyn muuttujan vaikutuksesta.

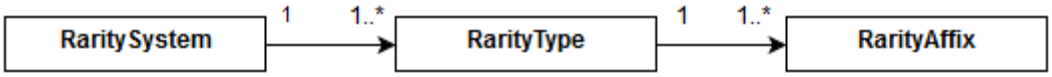
Efektit

Olento voi saada efektejä nauttimistaan kulutettavista tavaroista, jolloin ne ovat sen tilastoihin positiivisesti vaikuttavia, tai taioista, jolloin ne voivat olla myös negatiivisia, kuten palamista.

Efektit on tallennettu listaan, josta käydään jokaisella kierroksella ne läpi ja niiden vaikutus olennon tilastoihin suoritetaan, efektin ikää kasvatetaan ja iän tullessa täyteen poistetaan vaikuttava efekti olennon listasta. Efektien vaikuttaessa toistetaan myös jokaisella vuorolla niihin liitetty visuaalinen efekti. Samantyyppiset ja vaikutukseltaan samansuuruiset, mutta mahdollisesti eri ikäiset efektit liitetään yhteen niitä olennolle lisättäessä, jolloin efektien esittäminen käyttöliittymässä saadaan kompaktimmaksi.

4.9 Harvinaisuusjärjestelmä

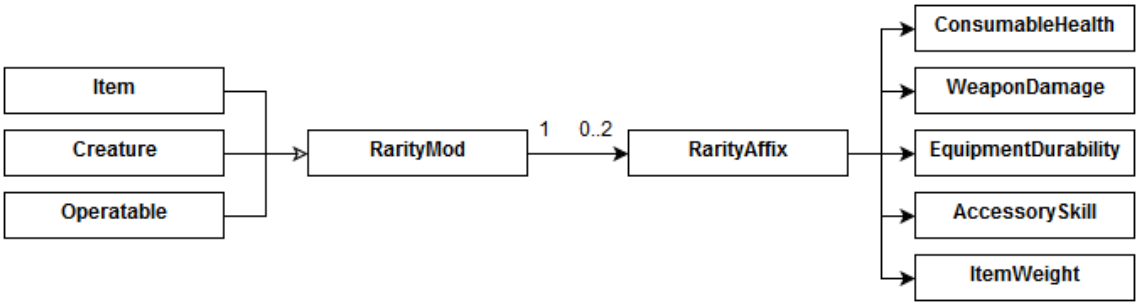
Kaikki mahdolliset harvinaisuus-affiksit generoidaan harvinaisuustyyppiin. Harvinaisuustyypissä on määritelty sen nimi, esiintyvyys, väri ja vaikutuskerroin objektien tilastoihin. Harvinaisuustyypit on tallennettu RaritySystem-luokkaan, josta sopivat affiksit haetaan objektia generoidessa sen esiintyvyyttä vastaavasta harvinaisuustyypistä. Harvinaisuustyyppien liittyminen harvinaisuusjärjestelmään havainnollistetaan kuvassa 11.



Kuva 11. Harvinaisuusjärjestelmän kuvaus. Tummat nuolet kuvaavat luokkien välisiä navigointisuuntia.

Simulaation jokaiseen dynaamiseen objektiin ja tavaraan liitettiin harvinaisuus perimällä RarityMod-luokka. Harvinaisuus vaikuttaa objektista riippuen eri tavalla. Olennoissa harvinaisuus vaikuttaa sen perustilastojen, kuten kestävyys ja taikavoima, suuruuksiin. Operoitavissa objekteissa harvinaisuus on vain visuaalinen lisä, poikkeuksena arkut, joissa harvinaisuus vaikuttaa myös arkusta löytyvien tavaroiden harvinaisuuteen. Laajin vaikutus harvinaisuudella on tavaroihin. Käyttöliittymässä harvinaisuus esitetään objektin värin muunnoksena ja affiksit objektin nimeen liitettyinä etu- ja jälkiliitteinä. Harvinaisuus-affiksin liittyminen peliobjektiin havainnollistetaan kuvassa 12.

Harvinaisuus-affiksit



Kuva 12. Kuvaus harvinaisuuden liittämisestä objektiin perinnän kautta. Tummat nuolet kuvaavat luokkien välisiä navigointisuuntia. Vaaleat nuolet kuvaavat luokan perintää.

RarityMod-luokka koostuu kahdesta harvinaisuus-affiksista. Tavaroihin liitetty harvinaisuus koostuu kahdesta affiksista. Jokaiselle tavaratyypille on määritelty omat harvinaisuus-affiksit. Ne ovat tavaran harvinaisuuteen liitettyjä ominaisuuksia, jotka vaikuttavat tavarasta riippuen sen tilastoihin, aseiden aiheuttamaan vahinkoon, painoon, tarkkuuteen, varusteiden puolustuskykyyn, kulutettavien tavaroiden parantamisvoimaan ja muihin efekteihin tai antavat koristavien tavaroiden kantajalle taikavoimia ja lisäävät niiden tehokkuutta.

4.10 Objektikirjasto ja objektitehdas

Objektikirjasto määriteltiin ObjectLibrary-luokassa. Luokka sisältää määritykset kaikista pelissä käytetyistä objekteista, taidoista ja malleista joiden mukaan olennot luodaan. Mallit laajennettiin Template-luokasta. Niiden mukaan olennot generoidaan alueille. Malleissa määriteltiin esimerkiksi olennoista niiden tilastot, raajat, omistamat tavarat, nimi ja niitä simulaation näkymässä kuvaava grafiikka.

Tehtävää generoidessa sen määrityksissä valittuja malleja käytetään luomaan simulaation alueelle mallien mukaisia kopioita olennoista käyttämällä objektitehdasta, joka määriteltiin ObjectFactory-luokassa. Tehtaalta objektien generointimetodeille annetaan parametrina objektin malli ja harvinaisuus. Luodun objektin tilastot generoidaan satunnaisuuden ja malleissa määriteltyjen arvojen perusteella. Parametrina annetun harvinaisuuden mukaan objektiin liitetään satunnaiset harvinaisuus-affiksit objektin tyypin mukaan.

4.11 Algoritmit

Pelissä käytettyjen useiden erilaisten algoritmien avulla oli mahdollista toteuttaa proseduraalinen pelimaailma. Alueiden generointiin kehitettiin omat generointialgoritmit, mutta muutoin esimerkiksi polun etsinnässä ja näkökentän laskennassa hyödynnettiin yleisesti peleissä käytettyjen algoritmien omia toteutuksia. Alkuun pelin toteutuksessa käytettiin libtcod-kirjaston tarjoamia algoritmeja polun etsinnässä, näkökentän mallintamisessa ja viivan piirrossa. Näistä algoritmeista polun etsintä ja näkökentän mallintaminen kuitenkin korvattiin omilla versiolla kehityksen myöhemmissä vaiheissa haluttujen ominaisuuksien puutteellisuuden ja suorituskyvyn heikkouden takia.

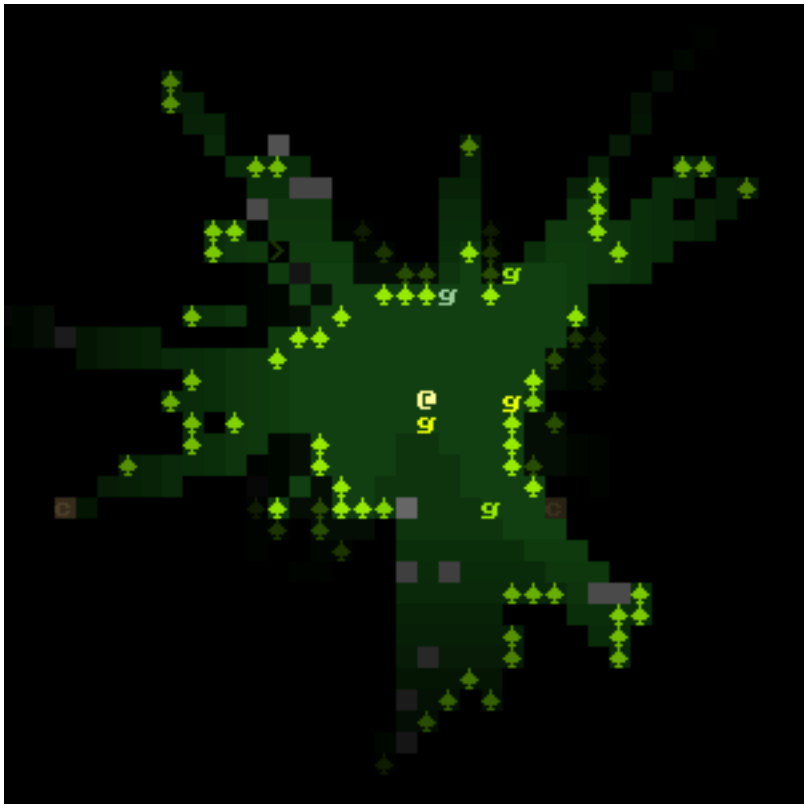
Näkökenttä

Pelin ensimmäisessä versiossa havaittiin libtcod-kirjastosta käyttöön otetun näkökentän laskennan algoritmin olevan epäsymmetrinen. Tällöin pelissä oliolle lasketussa näkökentässä havaittiin toinen olio, mutta tämän olennon näkökentässä ei havaitsevaa olentoa ollut. Tämä aiheutti tilanteita, joissa esimerkiksi pelaajan viholliset havaitsivat pelaajan, mutta pelaaja ei niitä.

Näkökentän mallinnus korjattiin ottamalla käyttöön niin kutsutun ”ray casting” -algoritmin oma toteutus. Algoritmin yleisenä periaatteena on piirtää suora säde katsojasta kaikkiin suuntiin, joihin katsojan oletetaan näkevän. Säteen viiva kuvastaa näkyvää osaa kentästä, ja sen piirto lopetetaan, jos eteen tulee objekti, jonka läpi ei katsojan haluta näkevän. Tällöin näkökenttä saadaan kaikkien säteiden piirtämien viivojen yhdistämisestä. Algoritmin laskentaan haluttiin myös mukaan näkökentän intensiteetin laskenta. Valmiin näkökentän esittäminen pelissä havainnollistetaan kuvassa 13.

Peliin toteutetussa algoritmin versiossa, joka määriteltiin AiFov-luokassa, katsojaa ympäröi rajatun suuruinen neliön muotoinen reunus. Reunuksen jokaiseen pisteeseen piirretään katsojasta suora viiva, jonka teho laskee suhteessa sen etenemään matkaan. Teho määrää näkökentän intensiteetin, joka kuvastaa näkymän läpinäkyvyyttä, jolloin näkökenttään saadaan mukaan häivytys. Toteutetun algoritmin puutteena voidaan pitää siinä esiintyvää ”gapping”-tapahtumaa, jossa algoritmi jättää katsojaan nähden tietyissä kulmissa oleviin suoriin pintoihin säteiden välille välejä, joita

katsoja ei näe. Myös yhden pisteen suuruisten kohteiden taakse voi muodostua vääränkokoinen katsojalle näkymätön alue tietyltä etäisyydeltä. Tämä tapahtuma johtuu siitä, että säteen viiva, jota piirretään, ei ole rajoitetussa koordinaatistossa tietyissä kulmissa täysin suora. Tällöin viivojen väliin syntyy alue, josta mikään reunalle piirretty viiva ei ole kulkenut. Mahdollinen korjaus tähän olisi useamman säteen käyttäminen, jolloin myös esimerkiksi reunapisteiden välisiin koordinaatteihin piirrettäisiin säde. Algoritmi on kuitenkin sellaisenaan nopea ja toteuttaa peliin vaaditun symmetrisyyden ja näkökentän intensiteetin.

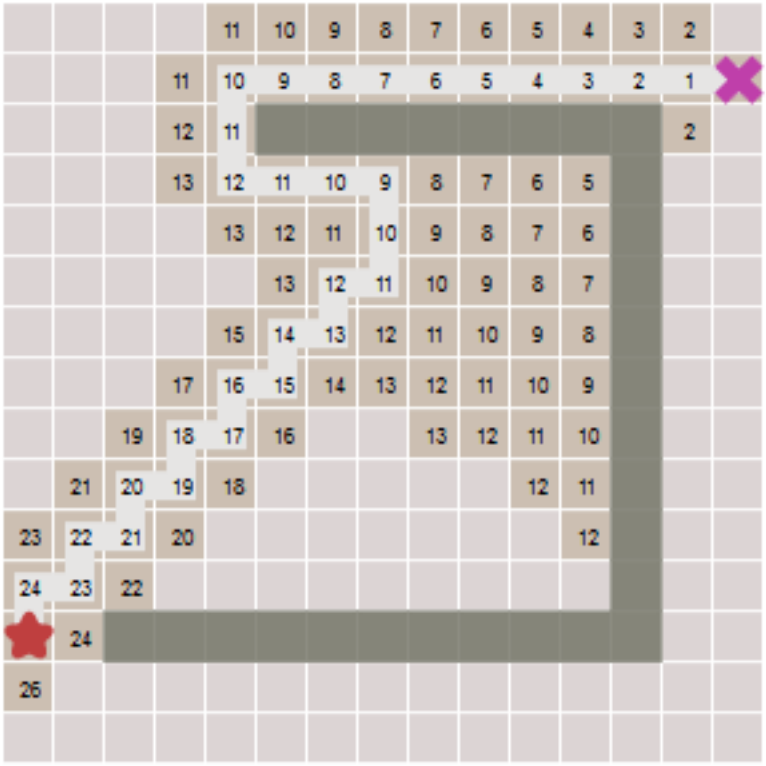


Kuva 13. Pelaajan ohjaaman olennon näkökenttä.

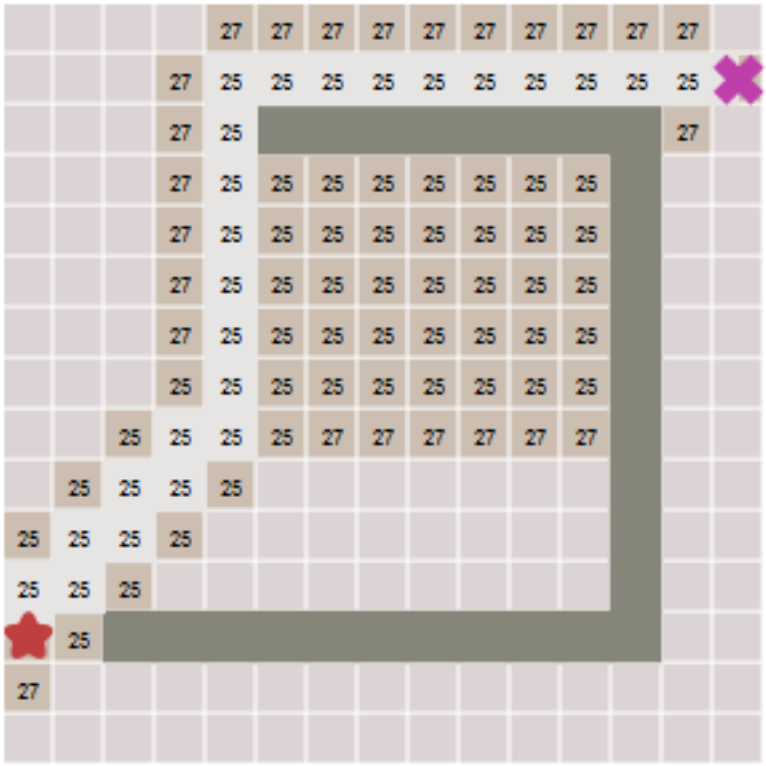
Polunetsintä

Polunetsintää käytetään simulaatiossa olioiden reittien etsinnässä, mutta myös pelin alueiden generoimisessa, kuten teiden ja purojen mallinnuksessa. Polunetsinnän algoritmi mahdollistaa reitin löytämisen kahden pisteen välillä pelin alueella, ottaen huomioon alueella sijaitsevat objektit, joiden läpi reitti ei voi kulkea ja objektit, joiden yli on helpompi kulkea.

Kehityksen alkuversioissa käytettiin libtcod-kirjaston tarjoamaa A*-algoritmia. Kirjaston algoritmin suorituskyky jäi kuitenkin heikoksi kun simulaatiossa oli useita polunetsintää käyttäviä olentoja, joten A*-algoritmista toteutettiin oma versio, jolla suorituskykyä saatiin parannettua. Toteutettu algoritmi käyttää heuristisena arvonaan etäisyyttä kohdepisteeseen joka on laskettu Chebyshev-etäisyyttä käyttäen. Chebyshev-etäisyys vastaa shakkiruudukossa käytettyjä etäisyyksiä. Heuristisen etäisyyden arvo kasvaa yhdellä jokaista ruutua kohden. Ruudun kautta kulkemisen hinnalla voidaan säätää algoritmin löytämää polkua. Esimerkiksi ruudun hinnan ollessa nolla, käytetään reitin määrittämiseen vain heuristista arvoa. Tätä hyödyntämällä tekoälyn ohjaamat olennot on saatu kulkemaan polkuja pitkin kun polun osan kautta kulkemisen hinnaksi on asetettu nolla. Antamalla heuristisen arvon vaikuttaa enemmän reitin muodostumiseen saadaan algoritmin suoritusta nopeutettua, mutta se myös huonontaa reitin optimaalisuutta eli löydetty reitti ei todennäköisesti ole lyhyin mahdollinen. Hinnan ollessa arvoa yksi suurempi, pyrkii algoritmi välttämään ruudun kautta kulkemisen, kunnes reitin pituus on kasvanut kautta kulkemisen hintaa suuremmaksi. Epäoptimaalinen reitti kohteeseen havainnollistetaan kuvassa 14. Kuvassa 15 havainnollistetaan optimaalinen reitti samaan kohteeseen.



Kuva 14. Kuvaus A*-algoritmin löytämästä reitistä kun heuristinen arvo vaikuttaa reitin määritykseen enemmän kuin ruudun kautta kulkemisen hinta [7].



Kuva 15. Kuvaus A*-algoritmin löytämästä reitistä kun heuristinen arvo on yhtäsuuri ruudun kauttakulkemisen hinnan kanssa [7].

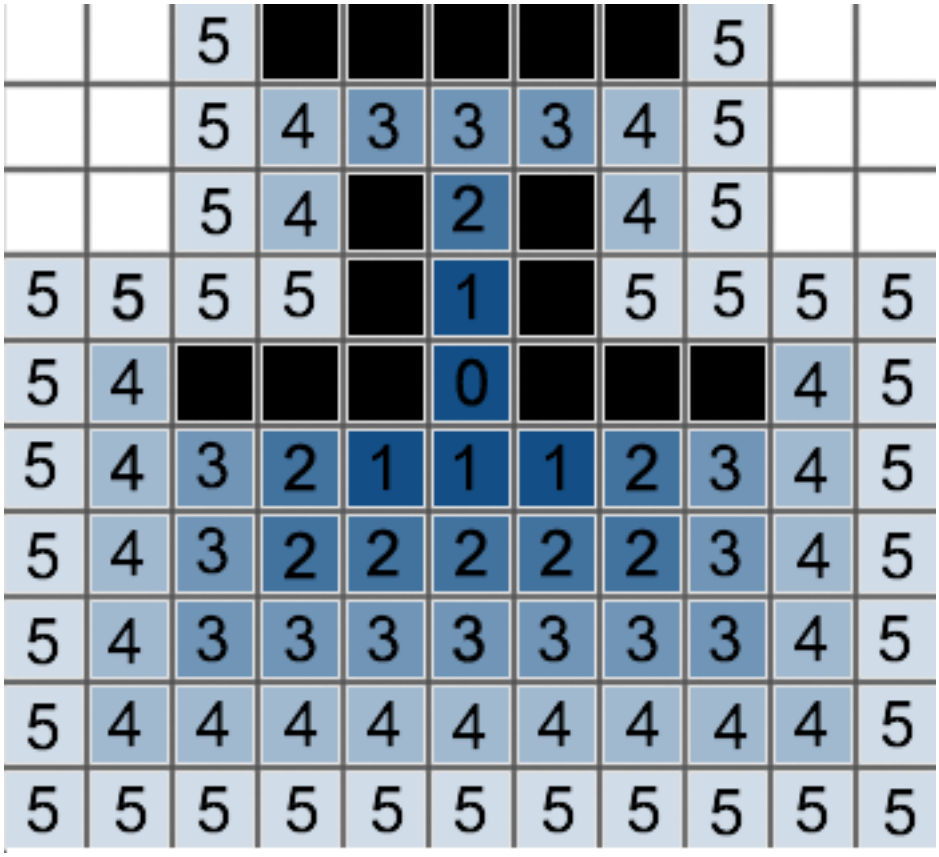
Toteutetulle algoritmille annetaan parametreina alku- ja kohdepiste sekä callback-funktio, jolla algoritmi hakee alueen maaston osille sen kautta kulkemisen hinnan. Algoritmi palauttaa koordinaateista koostuvan reitin, joka on validi reitti kohteeseen, jonka laskennassa on otettu huomioon maaston eri osien kautta kulkemisen hinta. Esimerkiksi jos olennon on raskaampi kulkea veden läpi, algoritmi pyrkii etsimään reitin, jolla veden voi kiertää. Palautettu reittiobjekti pitää muistissaan indeksiä, joka osoittaa sen nykyiseen koordinaattiin. Objektin rajapinnan kautta reittiä on mahdollista selata sekä eteen- että taaksepäin ja jäljellä oleva etäisyys reittiä pitkin nykyisestä pisteestä kohdepisteeseen voidaan hakea.

Flood fill

Flood fill -algoritmia on käytetty pelissä luomaan luolastoon vesistöjä. Algoritmilla voidaan skannata tietynkokoinen alue, jossa otetaan huomioon, että joitakin kohtia ei lasketa tulokseen mukaan. Algoritmi määriteltiin FloodFill-luokassa. Sitä kutsuttaessa sille annetaan parametreina skannattava alue, prosentuaalinen osuus, joka alueelta skannataan, ja callback-funktiot, joilla algoritmi hakee koordinaattien kautta kulkemisen mahdollisuuden ja tapahtuman, jota koordinaattien skannauksen jälkeen kutsutaan.

Esimerkiksi vettä generoitaessa callback-funktioiden kautta välitetään tieto siitä, että seinien läpi ei voida kulkea ja skannauksen jälkeen maaston osa asetetaan vedeksi.

Algoritmi toimii siten, että alkupisteestä tarkastetaan sitä ympäröivien koordinaattien kautta kulkemisen mahdollisuus. Jos koordinaattien läpi voidaan kulkea se lisätään skannaukseen, ja seuraavaksi juuri lisättyjä koordinaatteja ympäröiville koordinaateille toistetaan sama kutsu kuin alkupisteelle. Algoritmin lopputulos näyttää samalta kuin tietty määrä nestettä olisi kaadettu alueelle ja se on kiertänyt alueet, joihin neste ei pääse. Algoritmin lopputulos havainnollistetaan kuvassa 16.



Kuva 16. Flood fill -algoritmin kuvaus. Numero kuvaa monennellako kutsulla ruutu täytettiin. Mustat ruudut kuvaavat esteitä täyttöalueella.

4.12 Tekoäly

Pelin tekoälyt laajennettiin CreatureAi-luokassa. Tekoälyn avulla olennot päivittävät jokaisella vuorollaan näkökenttänsä ja määrittelevät seuraavan toimintonsa näkökentässä olevien objektien ja muiden muuttujien mukaan. Pelin eri olennoilla on

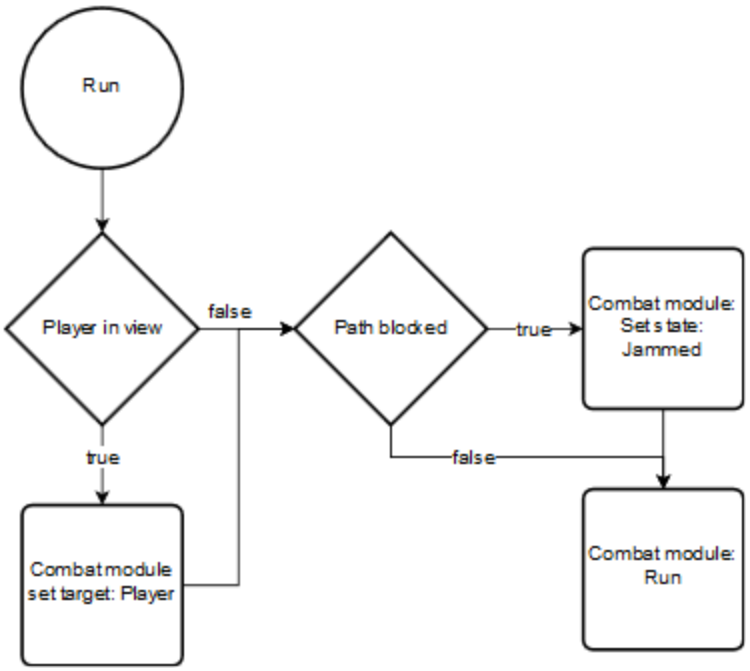
omat tekoälyt, joissa ne hyödyntävät tekoälymoduuleja seuraavan toimintonsa päättelemisessä. Tekoälyt toimivat tilakoneperiaatteella, jolloin niitä kutsuttaessa suoritetaan tekoälyn valitun tilan mukainen logiikka tekoälyn ohjaaman olennon seuraavaksi toiminnoksi. Pelaajan ohjaamalla olennolla tekoälyn kautta käsitellään ainoastaan näkökentän laskenta.

Esimerkiksi hirviö-tekoäly on jatkuvasti tilassa, jossa se kutsuu jokaisella kierroksella taistelumoduulia. Taistelumoduulia hyödyntää myös kyläläinen-tekoäly, mutta vain ollessaan taistelu-tilassa esimerkiksi pelaajan hyökätessä sitä kohti.

Tekoälyt

Hirviö

Hirviö-tekoäly on asetettu pelaajan vihollisille. Tekoäly etsii näkemistään olennoista pelaajaan olentoa ja sen nähdessään hyökkää sitä kohti. Tekoäly kutsuu taistelumoduulia jokaisella kierroksella. Jos näkökentässä oleva olento on pelaajan ohjaama olento se asetetaan taistelumoduulin kohteeksi. Tekoälyn logiikka havainnollistetaan kuvassa 17.



Kuva 17. Hirviö-tekoälyn yksinkertaistettu tilakone

Kyläläinen

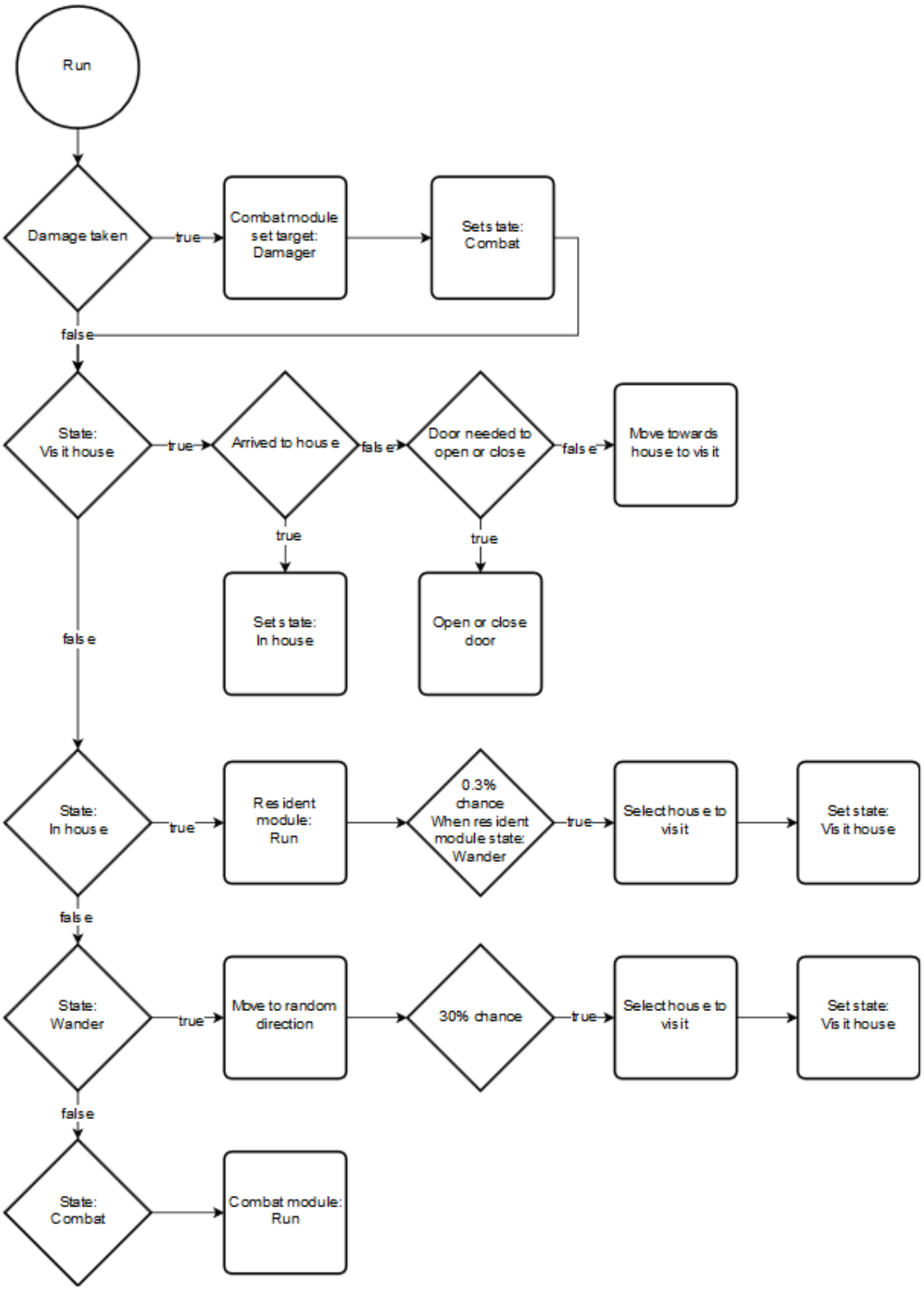
Kyläläinen-tekoäly mahdollistaa olentojen liikkumisen kyläalueella. Se on asetettu alueen olennoille, joiden halutaan vaeltelevan asuntojen välillä ja toimivan niissä. Tekoälyn tiloja ovat asunnossa vierailu, asunnossa oleminen, vaeltelu ja taistelu. Tekoälyn logiikka havainnollistetaan kuvassa 18.

Asunnossa vierailun tilassa lasketaan reitti asunnosta, jossa olento sillä hetkellä sijaitsee, satunnaiseen kylän asunnon keskipisteeseen. Reitin asetuksen jälkeen olento liikkuu reittiä pitkin asuntoon, kunnes on saavuttanut reitin määränpään, minkä jälkeen tila asetetaan asunnossa olemiseen. Liikkuessaan asunnosta toiseen tekoäly sulkee ja avaa ovet, jotka ovat reitillä. Jos olento törmää reitillä kulkiessaan toiseen dynaamiseen objektiin, vaihdetaan tila vaelteluun.

Asunnossa olemisen tilassa kutsutaan asukasmoduulia, jonka sisäisen tilan mukaan olennon liikkuminen asunnossa toteutetaan.

Vaeltelutilassa olento liikkuu satunnaiseen suuntaan, kunnes satunnaisuustarkistuksen toteutuessa lasketaan reitti satunnaiseen asuntoon ja tila vaihdetaan asunnossa vierailuun.

Taistelutila käynnistetään ja taistelumoduulia kutsutaan kohteeseen, joka on aiheuttanut vahinkoa tekoälyn ohjaamaan olentoon. Taistelutilassa taistelumoduulia kutsutaan sen sisäisen tilan mukaan.



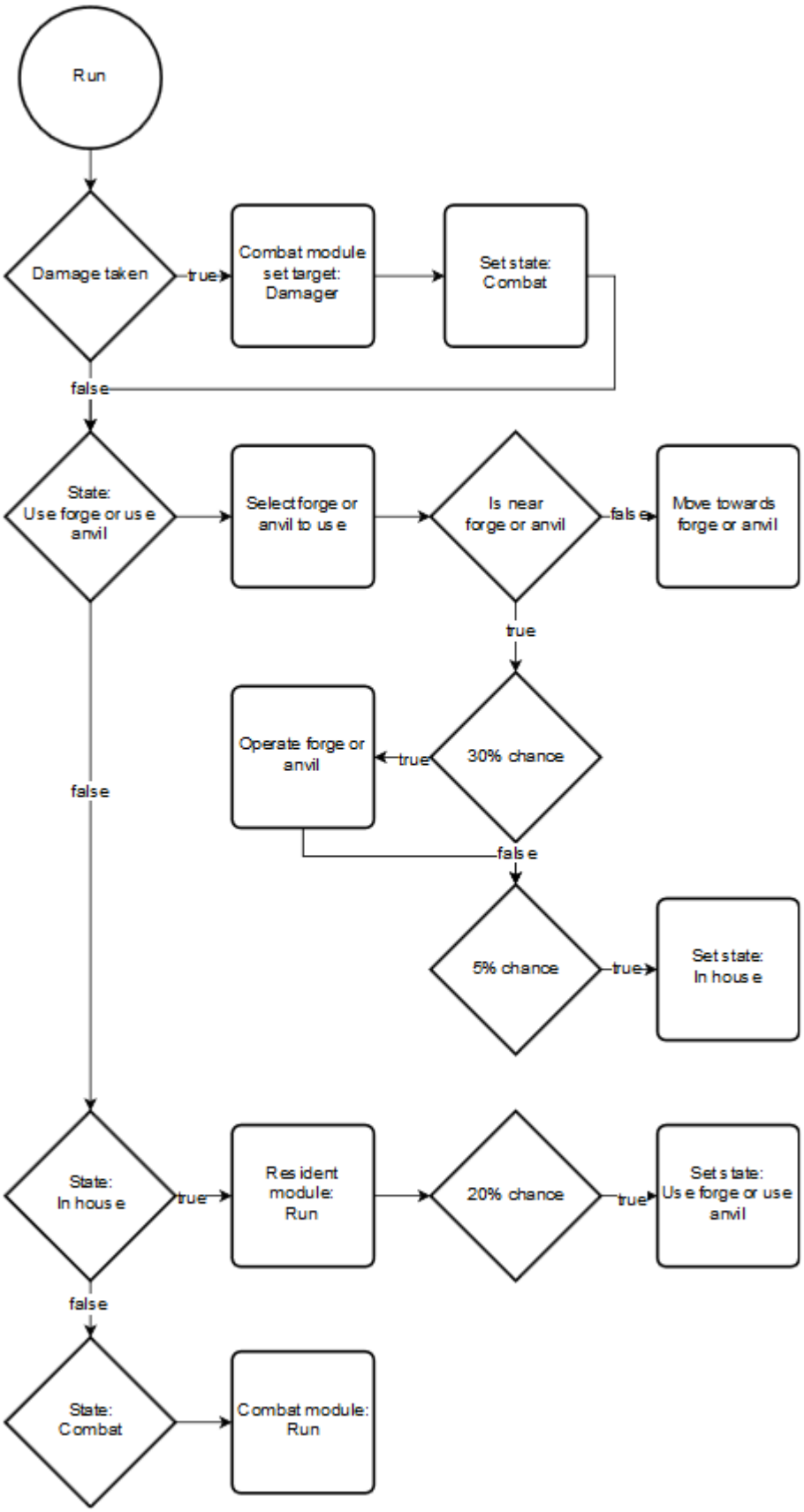
Kuva 18. Kyläläisen yksinkertaistettu tilakone

Seppä

Seppä-tekoäly on asetettu kylässä sijaitsevalle kauppiaalle, joka pysyttelee yhden asunnon sisällä ja operoi asunnossa sijaitsevia alasimia ja uuneja. Tekoälyn tiloja ovat asunnossa oleminen, alasimen käyttö, uunin käyttö ja taistelu. Tekoälyn logiikka havainnollistetaan kuvassa 19.

Asunnossa olemisen-tilassa kutsutaan asukasmoduulin sisäistä tilaa ja valitaan satunnaisuustarkistuksen toteutuessa seuraavaksi tilaksi uunin tai alasimen käyttö.

Alasimen ja uunin käytön tiloissa etsitään asunnosta olennon näkemä operoitava alasin tai uuni ja lasketaan reitti sen luokse. Olennon ollessa operoitavan objektin vieressä toistetaan visuaalista efektiä, joka kuvastaa kyseisen objektin käyttöä. Jos satunnaisuustarkistus toteutuu, seuraavaksi tilaksi vaihdetaan asunnossa oleminen.



Kuva 19. Seppä-tekoälyn yksinkertaistettu tilakone

Tekoäly-moduulit

Tekoälyn moduuli määriteltiin AiModule-luokassa. Siitä laajennetut moduulit toteuttavat run-metodin, jota kutsumalla moduulin logiikka suoritetaan. Moduulit toimivat tekoälyn tavoin tilakoneperiaatteella. Moduuli voi vaihtaa tilaansa sisäisesti toisen tilan tullessa päätökseen tai sitä käyttävän tekoälyn kutsun kautta.

Moduuleilla tekoälyn tila on voitu pilkkoa pienemmiksi kokonaisuuksia ja tekoälyille yhteisiä toimintoja on voitu jakaa niiden kesken moduulien kautta. Kukin tekoäly voi myös toteuttaa moduulien käyttämisen omalla tavallaan.

Taistelu-moduuli

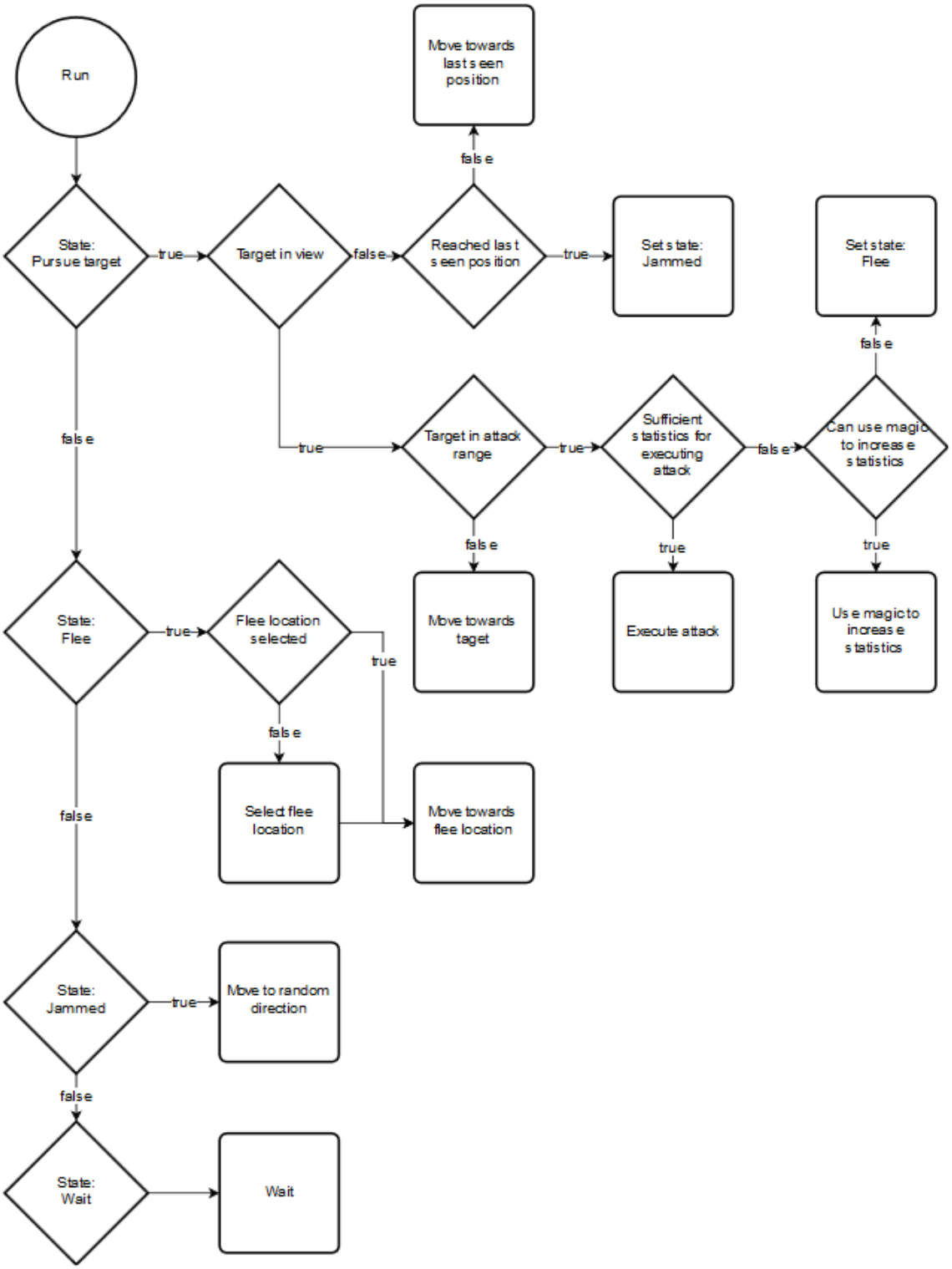
Taistelu moduulilla olennon tekoälyn on mahdollista valita sopivia toimintoja taistelun aikana. Moduulilla on neljä tilaa: odottaminen, kohteeseen eteneminen, jumiutuminen ja pakeneminen. Moduulia käytetään kutsumalla sen pursueAndAttack-metodia, joka asettaa uuden kohteen ja aloittaa sitä kohti hyökkäämisen. Moduulin logiikka havainnollistetaan kuvassa 20.

Odottamistilassa tekoälyn ohjaamalle olennolle ei ajetulla vuorolla anneta mitään toimintoa suoritettavaksi.

Kohteeseen etenemisen tilassa tarkistetaan, onko kohde edelleen näkökentässä. Jos kohde ei ole näkökentässä ja edellinen kohteeseen laskettu reitti on liikuttu loppuun, on kohde kadotettu ja moduulin sisäisesti siirrytään jumiutumistilaan. Hyökkäystoiminto kohdetta vastaan suoritetaan, jos kohde on näkökentässä ja moduulia käyttävän olennon on mahdollista sen tilastojen ja etäisyyden mukaan suorittaa jokin hyökkäystoiminto. Jos kohde on näkyvissä, mutta minkään toiminnon etäisyys ei ole riittävä, lasketaan reitti kohteeseen ja liikutaan sitä kohti. Moduulin tila vaihdetaan pakenemiseen, kun mitään olennon toimintoja ei ole mahdollista käyttää, esimerkiksi jos olennon jaksaminen on alhainen tai jos olento ei ole pukenut mitään asetta. Moduulin on myös mahdollista kohdetta vastaan hyökkäyksen sijasta kutsua tilastoja nostattavia taikoja ohjattavaan olioon, jos sen tilastot ovat alhaiset.

Jumiutumistilassa kohde on kadotettu ja olento siirtyy satunnaiseen suuntaan, kunnes kohde asetetaan moduulia ohjaavan tekoälyn kautta uudelleen.

Pakenemistilassa haetaan reitti lähellä sijaitsevaan vapaaseen ruutuun ja olento liikkuu sitä kohti, kunnes on saavuttanut valitun ruudun, minkä jälkeen olento jää odottamaan.

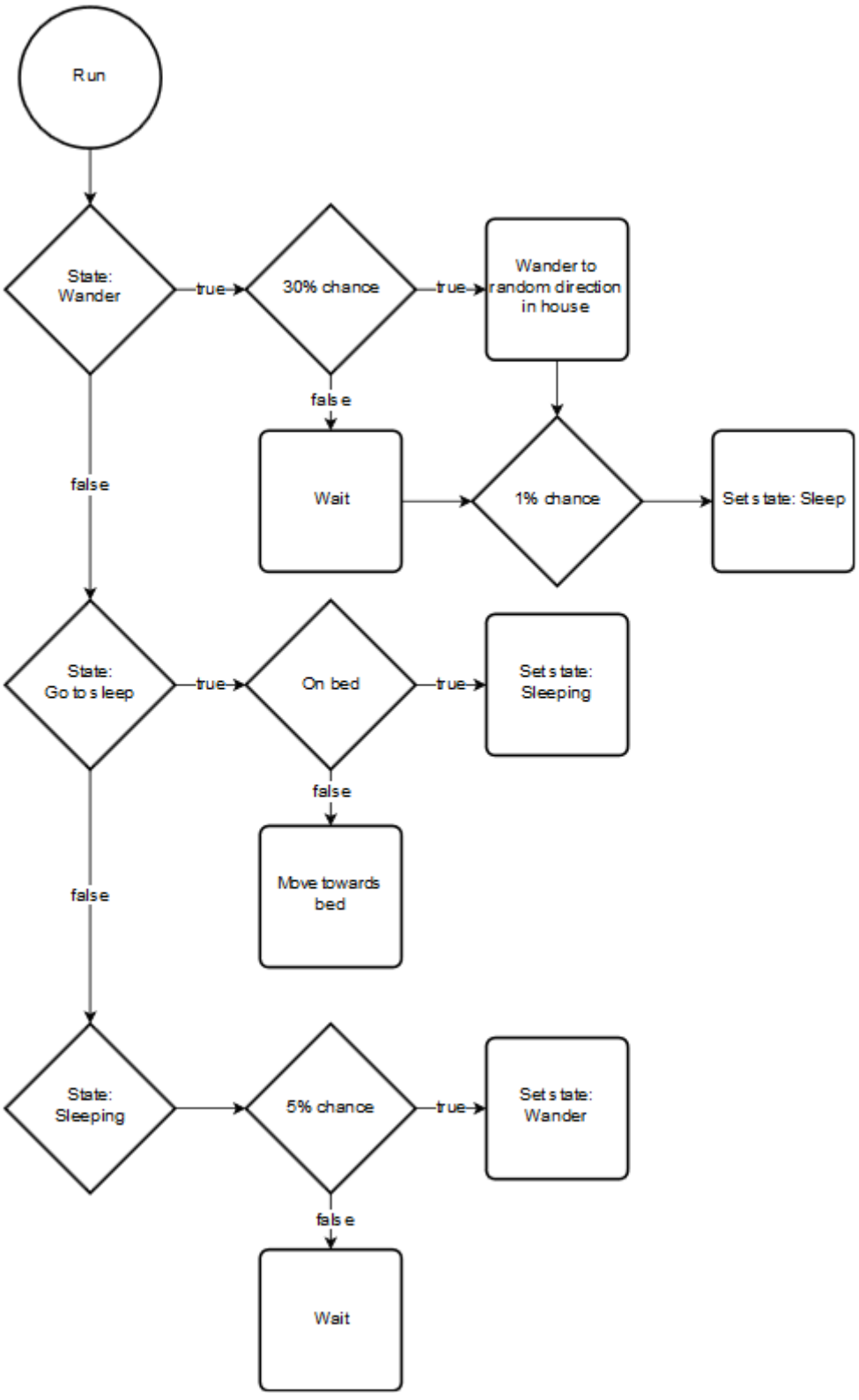


Kuva 20. Taistelu-moduulin yksinkertaistettu tilakone

Asukas-moduuli

Asukasmoduuli on laajennettu AiModule-luokasta. Moduulilla olennon tekoälyn on mahdollista valita sopivia toiminto asunnossa toimimiseen. Moduulin tiloja ovat: nukkuminen, nukkumaan meneminen ja vaeltelu. Moduulin toiminta aloitetaan asettamalla sille asunto, jossa olento sijaitsee, ja tila, josta halutaan aloittaa, ja lopuksi kutsumalla sen run-metodia. Moduulin logiikka havainnollistetaan kuvassa 21.

Vaeltelutilassa valitaan satunnaisesti olennon ympäriltä ruutu, joka sijaitsee asunnossa, ja olento siirtyy siihen. Jos määritelty satunnaisuus toteutuu, voidaan tila vaihtaa nukkumaan menemiseen, jolloin näkökentässä oleva vapaa sänky valitaan kohteeksi, jota kohti liikutaan. Kun sänky on saavutettu, asetetaan tila nukkumiseen, jonka aikana suoritetaan moduulia ajettaessa satunnaisuustarkistus sängystä poistumiselle ja asunnossa vaeltelun aloittamiselle.

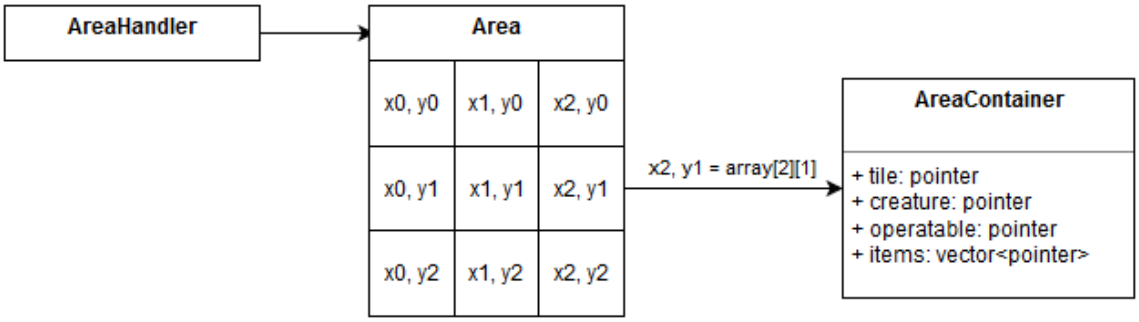


Kuva 21. Asukas-moduulin yksinkertaistettu tilakone

4.13 Alue

Kaikki pelin simulaatioon kuuluvat objektit sijaitsevat alueilla, jotka on laajennettu Area-luokasta, jolloin ne toteuttavat generate-metodin ja konstruktorin, jossa ne saavat generoinnissa käytettävät parametrit. Kulloinkin simuloitavaa aluetta vaihdetaan pelaajan liikkuessa alueelta toiselle. Alueella objekteja käsitellään niitä sijoittamalla, liikuttamalla, ja tuhoamalla. Alueelta voi myös tietyistä koordinaateista lähtien etsiä lähimmän kohdan, jossa haluttu maaston tyyppi tai avoin maaston osa sijaitsee. Objektien sijoittamismetodit käyttävät lähimmän vapaan alueen etsintää sisäisesti, koska näin voidaan varmistaa, että alueelle lisätty objekti siirtyy validiin kohtaan eikä esimerkiksi seinän sisään.

Alueen tilaa pidetään kaksiulotteisessa taulukossa, jossa jokaista koordinaattia vastaa yksi taulukon muistialue. Muistialueille on asetettu säilytystila-objektit, jotka on määritelty AreaContainer-luokassa. Niiden sisällä voidaan pitää useita tavaroita, yhtä oliota ja yhtä operoitavaa objektia. Säilytystila-objektien avulla samaan koordinaattiin voidaan tallentaa useampi eri objekti. Aluetta piirrettäessä säilytystiloista valitaan juoksevan indeksin mukaan tavara, joka alueen kohtaan piirretään. Näin pelaajalle voidaan vuorotellen esittää kaikki alueella sijaitsevat tavarat eikä esimerkiksi vain listan päällimmäisenä olevaa. Alueen rakenne havainnollistetaan kuvassa 22.



Kuva 22. Alueen rakenteen kuvaus. Taulukko esittää alueen ruutuja. Nuolet kuvaavat luokkien välisiä navigointisuuntia.

Kun alueelta haetaan x- ja y-koordinaatteja vastaava säilytystila, saadaan taulukosta suoraan osoittamalla koordinaatteja vastaavaan taulukon indeksiin. Alueelta hakeminen on tällöin nopeaa, verrattuna esimerkiksi lista-tietorakenteen läpi käymiseen oikeiden koordinaattien löytymiseen. Alueen dynaamiset objektit on kuitenkin tallennettu myös listaan, jos simulaation on tarve iteroida niiden kaikkien läpi.

Tämä taas on nopeampaa verrattuna siihen, että koko kaksiulotteinen taulukko käytäisiin koordinaatti kerrallaan läpi etsien kaikki dynaamiset objektit. Alueella on myös objektien yhteisesti käyttämä polunetsintämoduuli, jota käytetään alueiden generointiin ja tekoälyn ohjaamien objektien liikuttamiseen.

Alueita generoitaessa niille annetaan parametreina niiden koko, objektit, joita generointi käyttää, ja generointiin vaikuttavia parametreja, kuten luolaston monimutkaisuuteen tai avonaisuuteen vaikuttavia muuttujia. Alueille generoidaan aina maa ja ulkoreunan rajat, jotta olennot eivät pääse alueen rajojen ulkopuolelle ja riko simulaation mallinnusta.

Annettujen objektien generoinnissa sijoittamista helpottamaan luotiin luokkia, joiden käyttötarkoituksen mukaan objektit on ryhmitelty. Aluesijoitus on määritelty AreaDrop-luokassa. Sen määrityksiin asetetaan olennot ja operoitavat objektit sekä niihin liitetyt todennäköisyydet tulla generoiduksi ja alueen suuruus, jolle objektit sijoitetaan. Luokkaa on hyödynnetty alueen generoinnissa sijoittamaan satunnaisia määriä olentoja ja huonekaluja tai arkkuja alueen maastoon ja asuntoihin. Aluetta generoitaessa kutsutaan aluesijoituksen metodeja niihin alueen maaston kohtiin tai asuntoihin joihin objekteja halutaan generoida. Maastoon generoitaessa alueelta haetaan jokaiselle sijoitetulle objektille validi maaston osa määritellyn alueen sisältä ja objekti sijoitetaan siihen. Asuntoon generoitaessa operoitavat objektit asetellaan seinän viereen, muttei oven eteen ja olennon sijoitetaan huoneen keskelle.

Asunnot on määritelty AreaDen-luokassa. Asuntoon annetaan parametreina sen koon lisäksi sitä generoitaessa käytettävät materiaalit, kuten lattiat, seinät ja ovet. Lisäksi sille annetaan siihen sijoitettavat objektit antamalla sille parametrina aluesijoitusobjekti, jossa asuntoon sijoitettavat objektit ja niiden generoitumisen todennäköisyydet on määritelty. Asunto generoidaan sijoittamalla alueelle ensin asunnon lattiat, sitten seinät ja ovi sijoitetaan siten ettei ovi tule asunnon nurkkaan ja lopuksi alueenpudotusobjekti generoidaan asunnon sisälle, jolloin asunnon huonekalut ja asukkaat saadaan sopiville paikoille.

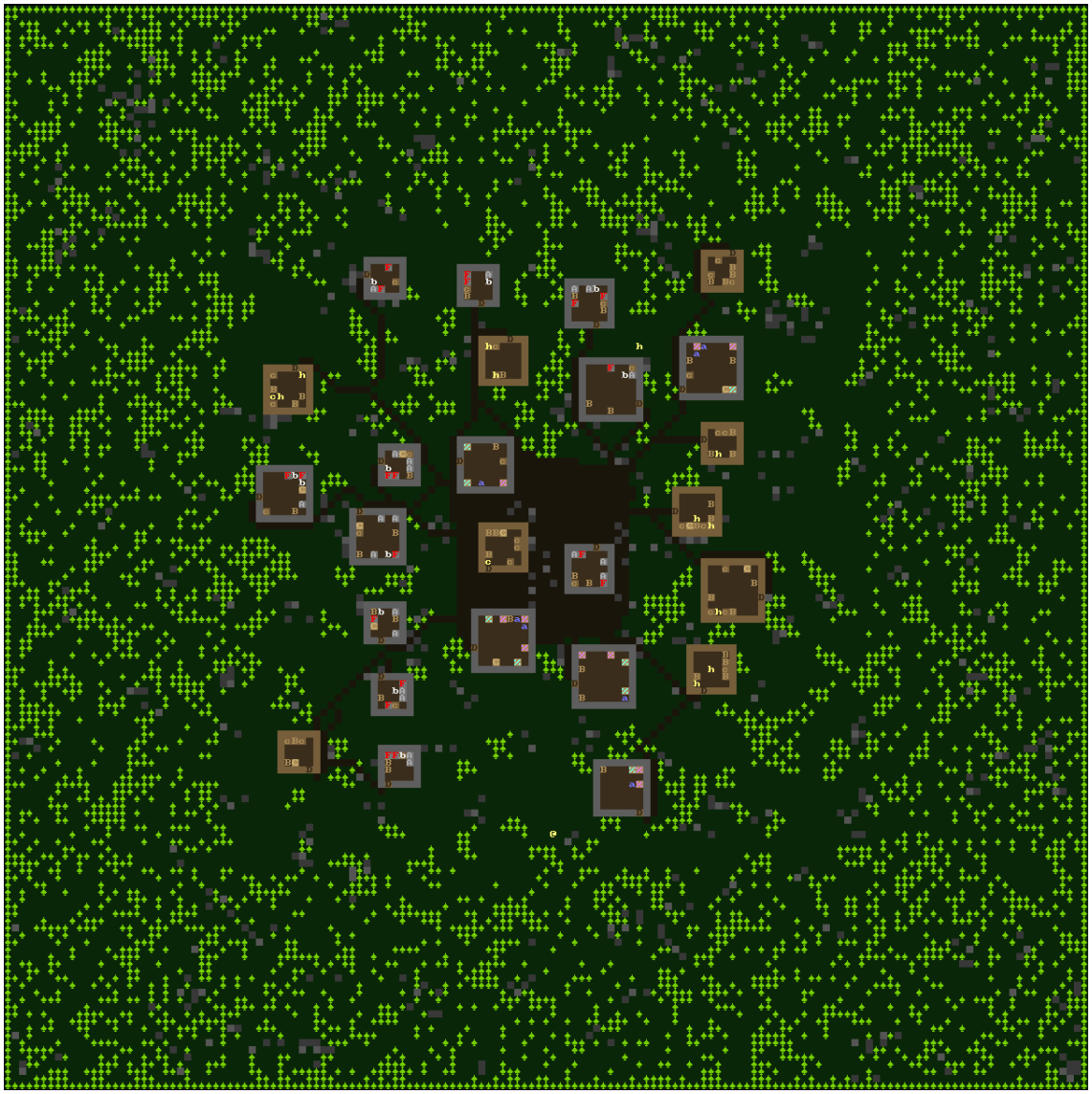
Kylä

Kylä on alueesta laajennettu pelialue, joka on pelin tehtävän määrityksissä pelaajan olennon lähtöpiste. Kun pelaajan olento on portaalin päällä, voi pelaaja palata kylään tai jatkaa seuraavalle alueelle Poikkeuksellisesti pelaajan ollessa kylässä voi sieltä

seuraavalle alueelle siirtyä mistä tahansa kohdasta. Kylä koostuu asunnoista ja teistä asuntojen välillä. Kylää reunustaa metsä, ja teistä muodostuu kylän keskelle aukio.

Kylässä on simuloitu pelaajan olentoa kohtaan neutraalisti käyttäytyviä tekoälyn ohjaamia olentoja. Joidenkin niiden kanssa pelaaja voi myydä löytämiään tavaroita ja ostaa uusia. Pelaajan on myös mahdollista hyökätä näitä olentoja vastaan jolloin niiden tekoäly pelaajaa kohtaan muuttuu negatiiviseksi. Olennot liikkuvat asuntojen välillä priorisoiden teiden käyttöä, ja asunnosta toiseen siirtyessään ne avaavat ja sulkevat asuntojen ovet. Ne myös näyttävät suorittavan päivittäisiä askareitaan, kuten nukkuvat, sepät takovat ja alkemistit keittävät juomia. Askareilla on pelin simulaatiossa vain visuaalinen vaikutus.

Kylälle annetaan parametreina maaston osat, joista se koostuu, ja asunnot ja niiden todennäköisyys tulla generoiduksi alueelle. Alueen maaston ja rajojen generoinnin jälkeen luodaan alueelle asunnot. Asunnot generoidaan siten, että alue jaetaan ristikkoon, jossa ristikon ruudut ovat erisuuruisia ja kuvaavat asuntojen tontteja. Jokaiselle ruudulle generoidaan asunnon todennäköisyysparametrin mukaan asunto tai se jätetään tyhjäksi. Sitten kaikkien asuntojen ovien välille generoidaan tiet käyttäen polunetsintäalgoritmia. Teitä generoitaessa polunetsintä suosii jo olemassa olevia teitä, joten tieverkosto haarautuu vähäisillä ylimääräisillä teillä kaikkien asuntojen välille. Seuraavaksi kylän ympärille generoidaan metsää, johon ripotellaan kiviä, ja kylän keskelle generoidaan teistä koostuva keskusaukio. Valmis kylä-alue esitetään kuvassa 23.



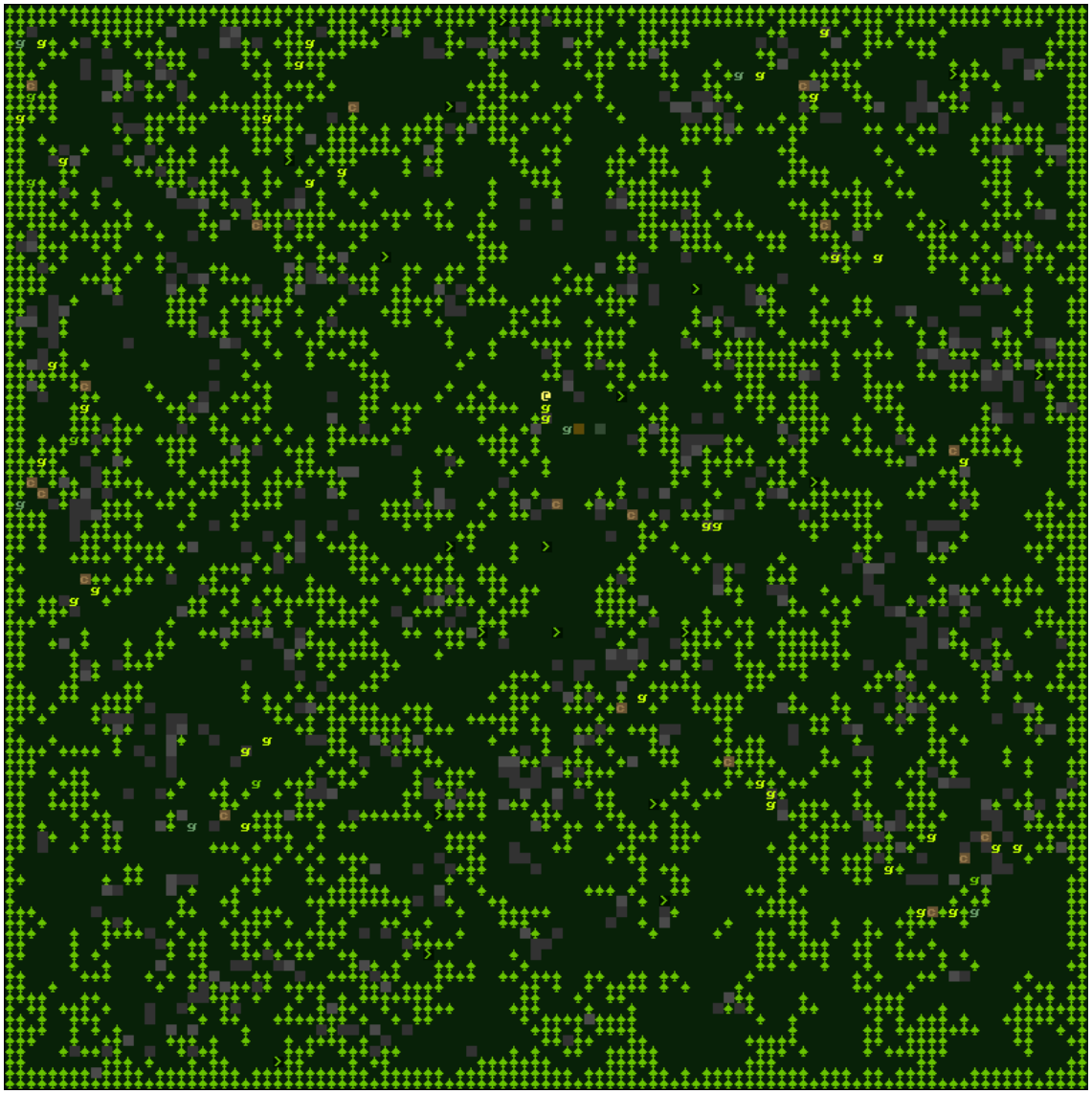
Kuva 23. Kylä-alue generoituna.

Metsä

Metsä on alueesta laajennettu pelialue, jonka generoinnissa käytetään puualueetta, joka on määritelty TreeSpot-luokassa. Puualue kuvaa puiden kasvua pisteestä hajautuen satunnaisiin suuntiin. Sille on määritelty sen generoimisessa käytetty puutyyppi, puiden kokonaismäärä, alueen kasvuhaarojen määrä ja maaston osat, joiden yli se ei voi kasvaa. Puualuetta generoitaessa lasketaan puiden määrä jokaista kasvuhaaraa kohden. Sitten jokaista haaraa kasvatetaan muiden määrän verran puualueen lähtöpisteestä satunnaiseen suuntaan. Haaran kasvatus lopetetaan ja toisen haaran kasvatus aloitetaan, jos kasvavan haaraan eteen tulee maaston osa,

jonka yli se ei voi kasvaa. Generoinnin lopputuloksena syntyy tähteä muistuttava alue puita.

Metsä saa parametreina prosentuaaliset puuston ja kivien osuuden arvot sekä puualueiden hajautuneisuuden, tiheyden ja aluesijoitusobjektin, jossa alueelle sijoitettavat objektit saadaan. Metsää generoitaessa lisätään ensin puualueet valitsemalla alueelta koordinaatteja, joista puualueiden kasvatus aloitetaan, ja sitten kasvattamalla ne algoritmin mukaan. Seuraavaksi alueelle lisätään kivet ja portaalit satunnaisiin kohtiin, ja lopuksi aluesijoitusobjektia kutsutaan myös satunnaisiin kohtiin, jolloin metsään saadaan halutut pelaajan vastustaja-olennot ja arkut. Valmis metsä-alue esitetään kuvassa 24.



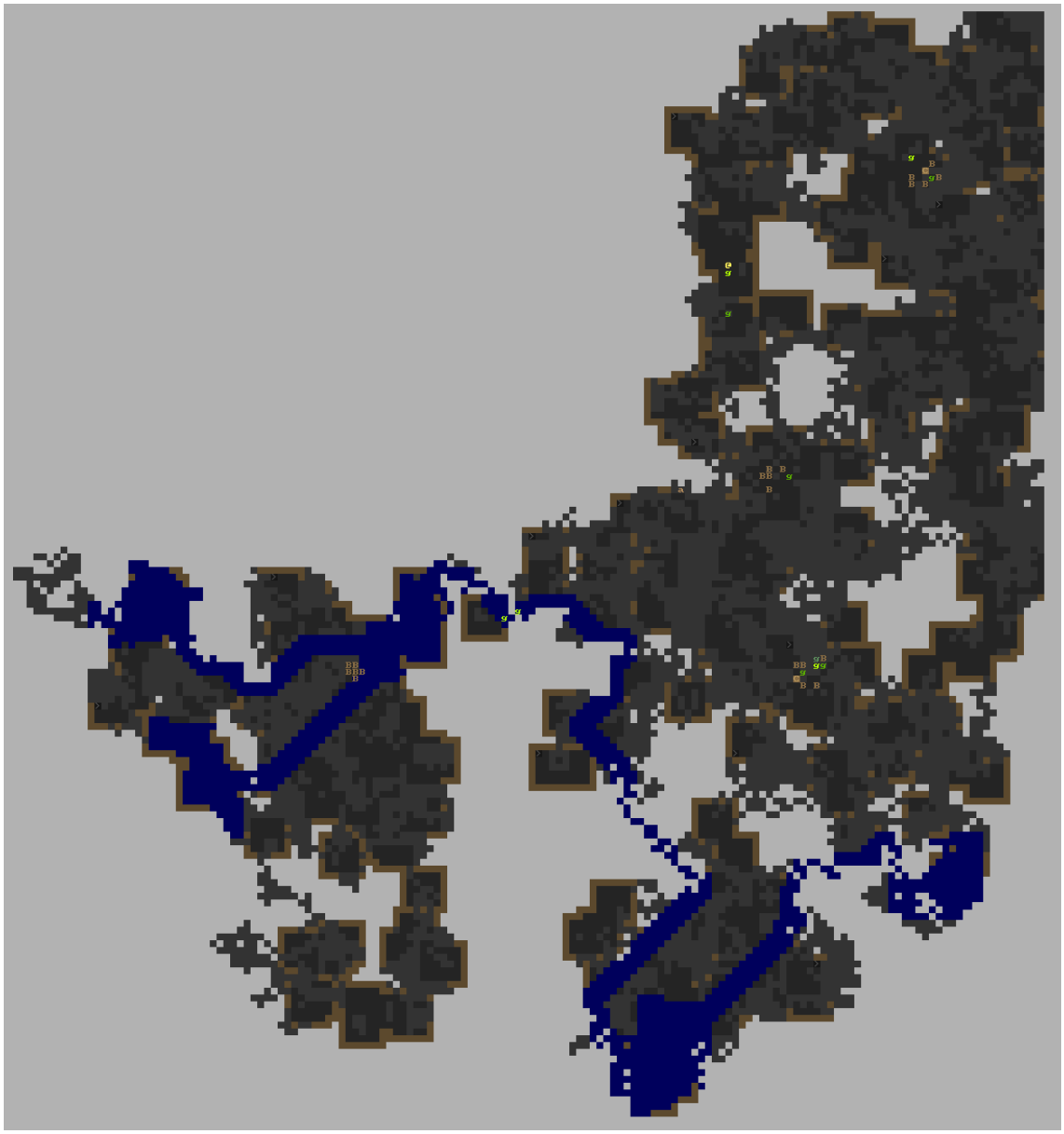
Kuva 24. Metsä-alue generoituna

Luolasto

Luolasto on alueesta laajennettu pelialue. Se saa parametreina luolaston seinien, maan ja veden maaston osat ja aluesijoitusobjektin, jota kutsutaan luolaston aukeille alueille, luolaston käytävien pituuden, aukeiden alueiden koon ja niiden esiintymistodennäköisyydet.

Luolasto generoidaan asettamalla alkuun luolan kaiverruskursori alueen keskipisteeseen. Luolastoa kaiverretaan siten, että sitä kaiverretaan, joko käytävän

pituuden verran satunnaiseen suuntaan tai luodaan aukea alue kursorin nykyiseen pisteeseen. Valittu toiminto valitaan määriteltyjen todennäköisyyksien mukaan. Jokaisen toiminnon jälkeen valitaan jälleen seuraavaa toiminto ja kaiverrusta jatketaan, kunnes määritelty prosentuaalinen osuus luolastoa on kaiverrettu. Jos aukea alue kaiverretaan, sijoitetaan sille alueelle todennäköisyyden mukaan myös aluesijoitusobjektin olennot ja operoitavat objektit. Kaiverruksen jälkeen alueelle sijoitetaan flood fill -algoritmia käyttäen vettä, josta muodostuu luolastoon lampia, ja niiden välille muodostetaan puroja käyttäen polunetsintäalgoritmia. Purojen muodostamisessa polunetsintä suosii jo olemassa olevaa vesistöä kulkemisessa, ja näin generoidut vesistöt vaikuttavat luonnollisemmin toisiinsa yhtyneiltä. Lopuksi portaalit sijoitetaan luolastoon satunnaisesti. Määrityksistä riippuen lopputuloksena voidaan saada aikaan luolasto, joka koostuu useista pitkistä ja kapeista käytävistä, joiden välillä esiintyy aukeampia alueita, ja sitä kattaa vesistö, jota pitkin pelaajan on helpompi suunnistaa. Valmis luolasto-alue esitetään kuvassa 25.



Kuva 25. Luolasto-alue generoituna

5 Yhteenveto

Insinöörityönä tehdystä pelistä saatiin kehitettyä toimiva versio, joka sisältää yhden tehtävän. Tehtävän sisältö on kuitenkin suppea, ja nykyinen versio on täten sopiva esittämään vain moottorin toimintaa. Myöskään varsinaista pelin vaikeuden tasapainotusta ei tehty sen enempää kuin että pystyttiin testaamaan tasapainottamisen

toteuttamisen helppous laajemman sisällön lisäämisen jälkeen globaalien tilastojen suhteiden muuttujien kautta.

Pelin käyttöliittymän käyttötilanteita ja arkkitehtuuria ei alkuun suunniteltu tarpeeksi tarkasti, joten käyttöliittymän muokkaaminen tukemaan toteutettuja ominaisuuksia osoittautui aikaa vieväksi. Käyttöliittymään toteutettiin useita kullekin kehykselle spesifejä erityistapauksia, ja joissakin kehyksissä toteutettiin samankaltaista koodia suorittamaan lähes samoja ominaisuuksia. Nämä voisi korvata toteuttamalla parempia käyttöliittymän yhteisiä komponentteja. Lisäksi harvinaisuuden lisääminen pelin objekteihin ilmeni haastavaksi ja aikaa vieväksi harvinaisuuden useiden eri objektista riippuvien vaikutustapojen vuoksi. Harvinaisuutta ja objektien tilastoja ja sen niihin vaikuttamista olisi mahdollista muokata geneerisemmäksi luomalla esimerkiksi tilastoista perintäluokkia, joihin harvinaisuus vaikuttaisi.

Toteutuksessa hyödynnetty libtcod-kirjasto osoittautui oletettua vähemmän tarpeelliseksi. Nykyisessä versiossa kirjasto toimii enää pääasiassa laitteistokiihdytetyn konsolin luomisessa, jossa se toimii hyvin ja tarjoaa kattavia vaihtoehtoja konsolin ruutujen manipulointiin. Kehityksessä käytetyt kirjaston algoritmit kuitenkin osoittautuivat vain suppeampiin käyttötarkoituksiin soveltuviksi.

Pelin moottorista saatiin kehitettyä tarpeeksi tehokas simuloimaan useita tekoälyn ohjaamia objekteja, joille kullekin simuloitaessa lasketaan näkökenttä, polunetsintä ja tekoälyn logiikan päivitys. Lisäksi laajojenkin pelialueiden generointi saatiin nopeaksi. Moottori on sellaisenaan hyödynnettävissä pelin jatkokehitykseen, tai sen erillisiä rakenteita ja osia on mahdollista hyödyntää uusissa projekteissa.

Lähteet

1 The Doryen Library. Verkkodokumentti. Jice, Mingos. <http://roguecentral.org/doryen/>. Luettu 11.4.2016.

2 What a roguelike is. 2013. Verkkodokumentti. Zapata, Santiago. <http://www.roguebasin.com/index.php?title=What_a_roguelike_is>. Muokattu 11.1.2013. Luettu 11.4.2016.

3 Davis, Ziff. 2007. Roguish Charm. Verkkodokumentti. <http://www.1up.com/features/essential-50-rogue>. Luettu 11.4.2016.

4 Wichman, Glenn R. 1997. A Brief History of "Rogue". Verkkodokumentti. <http://www.wichman.org/roguehistory.html>. Luettu 11.4.2016.

5 Wagner, James Au. 2000. The best game ever. Verkkodokumentti. <http://www.salon.com/2000/01/27/nethack/>. Luettu 11.4.2016.

6 Pitts, Russ. 2006. Secret Sauce: The Rise of Blizzard.Verkkodokumentti. <http://www.escapistmagazine.com/articles/view/video-games/issues/issue_48/289-Secret-Sauce-The-Rise-of-Blizzard>. Luettu 11.4.2016.

7 Patel, Amit. 2016. Introduction to A*. Verkkodokumentti <http://www.redblobgames.com/pathfinding/a-star/introduction.html>. Muokattu 22.2.2016. Luettu 17.4.2016.