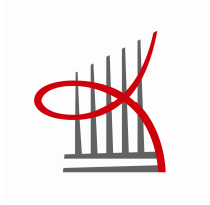


UGC-pelin toteutus HTML5-tekniikalla

Juho Vasenius

Opinnäytetyö
Joulukuu 2014
Tietojenkäsittely
Ohjelmistotuotanto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

JUHO VASENIUS
UGC-pelin toteutus HTML5-tekniikalla

Opinnäytetyö 50 sivua
Joulukuu 2014

Opinnäytetyön tavoitteena oli tutkia HTML5-tekniikan soveltuvuutta ja mahdollisuuksia pelien kehittämiseen Yleisradion lasten ja nuorten Internet-palveluiden tarpeiden näkökulmasta. Sen tarkoituksena oli kehittää ominaisuuksiltaan monipuolinen HTML5-peli joka painottaa luovaa ongelmanratkaisua ja käyttäjien tekemää sisältöä. Työ on tutkimusotteeltaan konstruktiivinen ja sen aineistoa kerättiin havainnoinnin, tekstianalyysin, keskustelujen ja käyttäjätestauksen keinoin.

HTML5 valikoitui pelissä käytettäväksi tekniikaksi sen laajan tuettavuuden ja helpon jakelun ansoista. Sama sovellus toimii kaikilla laitteilla, joille on saatavilla moderni web-selain. Monet käyttäjät välttävät ylimääräisiä sovellusasennuksia haittaohjelmien pelossa. Käyttäjän kynnys tutustua ilmaiseen HTML5:llä toteutettuun sovellukseen on pieni, sillä sen kokeileminen edellyttää vain linkin klikkaamista, eikä erillisiä asennuksia tarvita.

Opinnäytetyön tulos on *Kätköksi* nimetty peli ja sen toiminnallisuudet mahdollistava pelimoottori. Pelimoottori toteutettiin itse, jotta vältyttäisiin kompromisseilta pelin toiminnallisuuksissa ja mahdollistettaisiin sen hyödyntäminen myös toisissa ympäristöissä. Peli julkaistiin marraskuussa 2014 osana Yleisradion kouluikäisille lapsille suunnattua *Galaxi* tuotantoa. Peli on tällä hetkellä sivuston selkeästi suosituin sisältö.

Avainsanat: pelimoottori, html5, ugc

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in Business Information Systems
Option of software development

JUHO VASENIUS
UGC-game development with HTML5

Bachelor's thesis 50 pages
December 2014

Object of the thesis is to research the possibilities of the HTML5-technology for the needs of the Finnish Broadcasting Company's (Yle) office of internet-services for children and the young. Purpose is to develop a robust and rich HTML5 game which empathizes creative problem solving and user generated content. Thesis utilizes constructive research method and its material is collected by observing, text-analysis, conversations and user-testing.

One of the biggest strengths of HTML5 applications is their multi-platform nature. The same application works on desktop- and mobile-devices. Many users avoid installing small software in fear of malware. HTML5 applications require no installations and are therefore easy and safe to run.

The outcome of the thesis is a HTML5-game called *Kätkö* (a hiding place) and the game-engine made for it. The game-engine was made from scratch to avoid compromises in its features. *Kätkö* was released as part of Yle's production during the fall 2014 and it is at the moment the most popular content of the web-site.

Key words: game-engine, html5, ugc

SISÄLLYS

1 JOHDANTO 7
2 PELIN TEKNIIKAN TAUSTA 8
2.1 HTML5 8
2.2 JavaScript 9
2.2.1 Kielen tausta 9
2.2.2 Automaattinen roskienkeruu 9
2.2.3 Dynaaminen tyypitys 10
2.2.4 Objektit ja funktiot 10
2.2.5 Ohjelmointiparadigmat 11
2.2.6 Periyttäminen 12
2.3 Pelimoottorin yleiset tehtävät 12
2.4 Kolmannen osapuolen JavaScript pelimoottorit 14
3 PELIN ESITTELY 15
3.1 Pelin idea ja tavoite 15
3.2 Päävalikko 15
3.3 Kentän valinta ja pelaaminen 16
3.4 Kenttäeditori 18
3.5 Ohjeet 19
4 JÄRJESTELMÄARKKITEHTUURI 21
5 OHJELMISTOARKKITEHTUURI 22
5.1 Moduulipohjainen suunnittelumalli 22
5.2 Periyttäminen 23
5.3 Funktiokirjastot 24
5.4 Keskitetyt kuva- ja ääniresurssit 24
6 TIETOKANNAN RAKENNE 26
7 PELIMOOTTORIN TOTEUTUS 27
7.1 Törmäystenhallinta 27
7.2 Pelimaailman rakenne ja renderöinti 28
7.3 Valaistus 30
7.4 Käyttöliittymäkomponentit 32
7.4.1 Tekstikenttä ja fontti 32
7.4.2 Painike 34
7.4.3 Palette 35
7.5 Reitinetsintä 35
7.5.1 Algoritmin tausta 35
7.5.2 Esimerkki algoritmin toiminnasta 36

7.5.3 Pseudokoodi reitinetsinnästä 37
7.5.4 Toteutukseen valittu algoritmi 38
7.6 Reitinseuranta ja ballististen lentoratojen määrittäminen 39
7.7 Perlinin kohina 42
7.7.1 Alkuperä ja käyttökohteet 42
7.7.2 Esimerkki 44
8 KÄYTTÄJÄTESTAUS 46
8.1 Käyttäjätestauksen tausta 46
8.2 Hahmon liikuttaminen 46
8.3 Valikoissa liikkuminen 47
8.4 Oman kentän luominen 47
9 POHDINTA 49
LÄHTEET 50

LYHENTEET JA TERMIT

API	Application Programming Interface, ohjelmointirajapinta
CSS	WWW-dokumenteille kehitetty tyyliohjeiden laji
Collision detection	Peliobjektien törmäystenhallinta
HTML	Hypertekstin merkintäkieli
HTML5	Voi tarkoittaa HTML-merkintäkielen uutta versiota, yhdistelmää seuraavia web-tekniikoita: Javascript, CSS, HTML tai yleisnimitystä nykyaikaisille web-tekniikoille.
JavaScript	Pääasiassa web-ympäristössä käytettävä dynaaminen ohjelmointikieli
PHP	Erityisesti web-palvelinympäristössa käytettävä ohjelmointikieli
Renderöinti	Ohjelman piirto-operaatioiden prosessi
UGC	User Generated Content, käyttäjien luoma sisältö
W3C	World Wide Web Consortium. Kehittää ja ylläpitää WWW:n standardeja

1 JOHDANTO

Opinnäytetyön tavoite on tutkia HTML5-tekniikan soveltuvuutta ja mahdollisuuksia pelien kehittämiseen Yleisradion lasten ja nuorten Internet-palveluiden tarpeiden näkökulmasta. Tarkoitus on kehittää ominaisuuksiltaan monipuolinen HTML5-peli ja rakentaa samalla tehokas pelimoottori, jota voidaan käyttää jatkossa muissakin pelitoteutuksissa. Opinnäytetyön tutkimusote on konstruktiivinen ja sen aineistoa kerättiin havainnoinnin, tekstianalyysin, keskustelujen ja käyttäjätestauksen keinoin. *Kätköksi* nimetty peli julkaistiin osana Yleisradion lasten- ja nuorten internetpalveluiden tuotantoa marraskuussa 2014. *Kätkö* on tällä hetkellä *Galaxin* verkkopalvelun selkeästi suosituin sisältö.

Ylen lasten- ja nuorten Internet-palvelut tuottavat useita palveluita, joihin kuuluvat muiden muassa *Pikku Kakkosen* ja *Galaxin* Internet-palvelut. Nämä palvelut koostuvat monimediallisesta sisällöstä, jonka piiriin kuuluvat myös pelit. Näitä pelejä on aikaisemmin toteutettu pääasiallisesti Adoben Flash-tekniikalla. Kyseisen tekniikan tuettavuus on kuitenkin vähenemässä ja erityisesti mobiilikäyttäjien tavoittaminen Flash:lla on vaikeaa. On myös epävarmaa aikooko Adobe jatkaa Flash-kehitysympäristön ja sen käyttöjärjestelmäriippumattoman ajoympäristön AIR:n kehitystä tulevina vuosina. Flash:n syrjäyttäjäksi ovat nousseet ns. HTML5-tekniikat. Tässä yhteydessä HTML5:llä tarkoitetaan selaimessa suoritettavien sovellusten kehittämistä web-tekniikoilla eli pääasiallisesti Javascriptillä, CSS:llä ja HTML:llä.

Olen kerännyt pelimoottorin ohjelmoimiseen tarvittavia tietoja ja taitoja toteuttamalla useita kymmeniä pienempiä Flash- ja HTML5-pelejä ja sovelluksia. Lukuisia näitä Flash:lla toteutettuja pieniä pelejä on julkaistu laajemman pelikokonaisuuden yhteydessä (http://yle.fi/bofori/) vuosien 2012 – 2014 aikana. Näistä kokemuksista on ollut hyötyä yleisen ohjelmointitaidon kehittymisessä ja pelimoottorin arkkitehtuurin sekä palvelintoiminnallisuuksien toteuttamisessa.

2 PELIN TEKNIIKAN TAUSTA

2.1 HTML5

Puhuttaessa HTML5:stä, tarkoitetaan toisinaan hieman eri asioita. HTML5-kieli on uusittu versio verkkosivujen kehittämiseen käytetystä HTML-merkintäkielestä. HTML5 julkaistiin virallisesti 28. lokakuuta 2014 ja siitä tuli W3C:n suositus. Uudistus pitää sisällään mm. uudet `<canvas>`- `<video>`- ja `<audio>`-elementit, sekä tarkan kuvauksen HTML-dokumentin rakenteesta – dokumenttioliomallista. W3C on järjestö, joka kehittää ja ylläpitää standardeja maailmanlaajuiselle Internetille. W3C:n suositus on huolellisesti laadittu ja testattu standardi, joka on valmis implementoitavaksi yleisesti. Sen käyttöönottoa suositellaan kaikille sen ongelmakentän parissa työskenteleville. (World Wide Web Consortium, 2014.)

Puhuttaessa pelkästä HTML5:stä, viitataan yleensä laajemmin nykyaikaisiin webtekniikoihin. HTML5-kielen uusien ominaisuuksien lisäksi näitä ovat CSS tyyliohjeiden uudet ominaisuudet sekä sovellusrajapinnat, kuten Web Audio API, WebGL tai Geolocation API. HTML5:llä ei tällöin ole tarkkaa määritelmää, vaan se kattaa kaikki toiminnallisuudet, jotka toimitetaan loppukäyttäjälle modernin web-selaimen kautta ja jotka eivät tarvitse erillisiä asennuksia.

HTML5-sovelluskehitys on sovellusten toteuttamista webin avoimilla tekniikoilla. Ohjelmointikielenä käytetään tällöin JavaScriptiä. Web-selaimet tarjoavat JavaScriptin ajamiseen tarvittavan virtuaalikoneen, joka tulkitsee ja suorittaa ohjelmakoodia. Selaimet tarjoavat virtuaalikoneelle rajapinnan, jonka kautta ohjelmistokehittäjä pääsee käsiksi edellä mainittuihin ominaisuuksiin: HTML-dokumenttioliomalliin, CSS muotoiluihin ja muihin sovellusrajapintoihin.

2.2 JavaScript

2.2.1 Kielen tausta

JavaScript on dynaaminen ohjelmointikieli, jonka ensisijainen käyttökohde ovat web-selaimet. Kielen kehitti alun perin *Netscape Communications Corporation* mahdollistamaan käyttäjän koneella suoritettava, web-sivun toimintaan vaikuttavan ohjelmakoodi. Kieli on nykyisessä muodossaan dynaamisesti tyypitetty, automaattista roskienkeruuta käyttävä tulkattava oliopohjainen komentosarjakieli. Sen syntaksi mukailee löyhästi C-ohjelmointikieltä. (Mozilla Developer Network, 2014.)

JavaScriptiä käytetään perinteisesti responsiivisten web-sivujen luontiin, mutta sillä voidaan tehdä myös pelejä, palvelinsovelluksia sekä työpöytä- ja mobiilisovelluksia. JavaScriptin sanotaan olevan näennäisen helppo kieli, jonka kanssa on kuitenkin helppoa hankkiutua vaikeuksiin, mikäli sen sisäistä toimintaa ei tunne.

2.2.2 Automaattinen roskienkeruu

Automaattinen roskienkeruu tarkoittaa sitä, että ajoympäristö huolehtii itse muistinhallinnasta. Kehittäjä ei tällöin määritä itse käyttämiensä objektien elinkaarta niiden tuhoamisen osalta, eikä voi suoraan kontrolloida tiettyä muistialuetta. Automaattinen roskienkeruu toimii ajettavan sovelluksen taustalla etsien sovelluksen muistialueelta tietoja, joihin ohjelma ei tule enää viittaamaan ja merkitsee ne sallituiksi ylikirjoittaa.

Automaattisessa roskienkeruussa on hyvät ja huonot puolensa. Se yksinkertaistaa ohjelmointia ja estää osaltaan muistivuotoja, jotka voivat johtaa tietokoneen käyttömuistin ylikuormittumiseen. Se kuitenkin myös mahdollistaa sellaisen muistia vuotavan koodin osittaisen toiminnan, joka paljastaisi todellisen luontonsa selkeämmin, mikäli automaattista roskienkeruuta ei käytettäisi.

Edellä mainittu tilanne voi aiheutua esimerkiksi, mikäli koodi luo jatkuvasti suuren määrän uusia objekteja, jotka pian käyttönsä jälkeen hylätään. Automaattinen roskienkeruu saattaa tällöin riittää muistivuodon pitämiseen kurissa, mutta ylityöllistettynä ai-

heuttaa ongelmia sovelluksen toiminnalle. Tämä voi näkyä käyttäjälle sovelluksen hetkittäisinä pysähtymisinä, pätkimisenä tai lopulta kaatumisena.

2.2.3 Dynaaminen tyypitys

JavaScript on dynaamisesti tyypitetty kieli. Se tarkoittaa sitä, että koodissa määritetyt muuttujat eivät ole sidottuja tiettyyn tietotyyppiin. Sama muuttuja voi ensin viitata desimaalilukuun ja myöhemmin objektiin. Myös dynaaminen tyypitys tuo mukanaan hyviä ja huonoja seurauksia. Hyvää on muuttujien nopea määrittäminen ja tyyppimuunnosten automaattisuus, mutta samalla dynaaminen tyypitys mahdollistaa vaikeasti löydettävien virheiden tekemisen ja on osaltaan syynä heikommalle suorituskyvylle, sillä tietokoneen täytyy tulkita muuttujien tietotyyppejä ajonaikaisesti.

Dynaaminen tyypitys tekee koodista myös toisinaan vaikeaselkoista. Mikäli muuttujaan ei sitä esiteltäessä alusteta mitään arvoa, ei esittely kerro mitään muuttujan tietotyypistä. Myöskään parametreja käyttävän funktion esittely ei välttämättä kerro mitään sen edellyttämien parametrien tietotyypeistä. Tällöin koodin ulkoasun selkeys, testaus ja dokumentointi painottuvat.

2.2.4 Objektit ja funktiot

Lähes kaikki elementit JavaScriptissä, joko ovat objekteja tai käyttäytyvät kuten objektit. JavaScriptin objektit ovat assosiatiivisia taulukoita jotka sisältävät avain-arvo -pareja. (Lindley 2012, 15.) Myös kielen funktiot ovat objekteja ja niitä voi käyttää muiden objektien tapaan toisten funktioiden parametreina ja palautusarvoina. Funktioilla on kaksoisrooli, sillä ne toimivat *new* notaation avulla objektien konstruktoreina.

JavaScript sallii sisäkkäiset funktiot. Sisäfunktion sanotaan muodostavan ympäröivän funktionsa kanssa sulkeuman, jolloin ulkofunktion lokaalit muuttujat yhdessä sen parametrien kanssa tulevat sisäfunktion käyttöön ja säilyvät, vaikka ulkofunktio palautuisi. Sulkeumaa käytetään suositussa moduulipohjaisessa suunnittelumallissa mahdollistamaan kapselointi eli objektin yksityisten ja julkisten ominaisuuksien erottaminen. Sulkeuman avulla voidaan myös jäljitellä nimiavaruuksien toimintaa.

2.2.5 Ohjelmointiparadigmat

JavaScript tukee erilaisia ohjelmointiparadigmoja, kuten funktionaalista, strukturoitua, ja oliopohjaista ohjelmointiparadigmaa (Mozilla Developer Network, 2014). Funktionaalisen paradigma edellyttää edellä mainittua funktioiden käsittelyä tarvittaessa toisten funktioiden parametreina ja palautusarvoina. Funktionaaliselle ohjelmoinnille on ominaista, että ohjelman rakenne koostuu funktioista, jotka ovat itsenäisiä kokonaisuuksia. Tällaiset funktiot saavat parametreinaan kaiken tiedon, jonka ne tarvitsevat toimiakseen. Ne eivät tarvitse suorituksensa aikana mitään tilatietoja muilta ohjelman osilta. Tällainen rakenne tuo tiettyjä etuja. Samojen parametrien voidaan esimerkiksi olettaa aina johtavan samaan palautusarvoon. Tällöin voidaan ylläpitää tietojärjestelmää tietylle funktiolle annetuista parametreista ja niiden vasteista, jolloin samaa laskentatyötä ei tarvitse tehdä kahdesti. Funktionaalinen ohjelmointi voi myös auttaa löytämään toimintoja, jotka voidaan toteuttaa rinnakkaisesti. Jos funktiokutsut eivät vaikuta toistensa parametreihin, eivätkä kommunikoi niiden ulkopuolisten muuttujien kanssa, ne voidaan ajaa rinnakkaisesti.

Strukturoitu ohjelmointiparadigma tarkoittaa ohjelman rakentamista hyödyntäen monipuolisesti erilaisia tietyn kielen tarjoamia kontrollirakententeita, kuten erilaisia silmukoita ja ehtorakenteita. Tarkoituksena on välttää koodin liiallista monimutkaistumista, joka olisi väistämätöntä, mikäli käytettäisiin ainoastaan ehdollisia GOTO komentoja.

JavaScriptillä toteutetaan olio-ohjelmointia, mutta sen toteutus poikkeaa useista muista oliopohjaisista kielistä. JavaScriptissä ei ole luokkia eikä objekteille ei voi suoraan määrittää julkisia ja yksityisiä funktioita tai muuttujia, eli tehdä kapselointia. Näitä ja muita olio-ohjelmoinnin ominaisuuksia voidaan kuitenkin jäljitellä noudattamalla sopivia suunnittelumalleja. Koska JavaScriptissä ei ole luokkia, myös periyttäminen toteutetaan monista olio-ohjelmointiin suunnitelluista kielistä poikkeavalla tavalla.

2.2.6 Periyttäminen

Periyttäminen toteutetaan JavaScriptissä natiivisti objektien prototype ominaisuuden avulla, mutta sovelluskehittäjä voi valita toteuttavansa sitä omassa koodissaan myös muilla suunnittelumalleilla. Prototype periyttämistä kuvaillaan usein jokseenkin vaikeaselkoiseksi. Yksinkertaistettuna se perustuu tietyllä tapaa luotujen objektien automaattisesti saamille ominaisuuksille, jotka linkittyvät toisiinsa ns. prototype ketjussa.

Mikäli objektilta yritetään aksessoida jotakin ominaisuutta, jota siltä ei löydy, ominaisuutta yritetään seuraavaksi etsiä objektin prototypestä ja tämän jälkeen objektin prototypen konstruktorin prototypestä ns. prototype-ketjua pitkin, mikäli tällainen rakenne on luotu. Lindley (2012, 100) suosittelee prototype -periyttämisen käyttöä tai vähintään siihen tutustumista, sillä katsoo sen olevan edellytys kielen syvemmälle ymmärtämiselle.

2.3 Pelimoottorin yleiset tehtävät

Pelimoottori on ohjelmistokehys, joka mahdollistaa pelin toiminnallisuudet. Sen tehtäviä ovat muun muassa pelin renderöinti, fysiikanmallinnus, osumatestaus, käyttäjän syötteisiin reagointi, animointi, äänten toistaminen ja pelihahmojen tekoäly. Riippuu käsitteenmäärittelystä, ajatellaanko kaikista peleistä löytyvän pelimoottori. Erityisesti pienten pelien kohdalla pelimoottorin ja tietylle pelitoteutukselle spesifin toiminnallisuuden raja on häilyvä.

Yksinkertaiselle pelimoottorille on ominaista, että sen toiminnallisuudet rakentuvat yhden toistorakenteen ympärille. Pelimoottorin ytimessä on tällöin funktio, joka huolehtii peliobjektien sijaintitietojen päivittämisestä, käyttäjän syötteiden käsittelystä ja renderöinnistä yhden ohjelmakierron osalta. Pelimoottori mahdollistaa monipuolisempia toiminnallisuuksia tarjoamalla kehittäjän käyttöön rajapintoja erilaisiin palveluihin. Nämä palvelut voivat sisältää vaikkapa rajapinnan tiedustella pelin tilaa. Tilat voivat kertoa esimerkiksi auki olevista valikoista ja *peli käynnissä* tai *peli ohi* tilasta. Palvelut voivat sisältää myös esimerkiksi keinon luoda uusia peliobjekteja, toistaa äänitiedostoja tai vaikkapa tilata reitin kahden pisteen välillä pelimaailman sokkelossa.

Perinteisesti pelit, jotka sisältävät liikkuvia peliobjekteja tarvitsevat keinon laskea niiden sijainteja tietyin aikavälein. Tasainen liike voidaan tällöin toteuttaa lisäämällä objektin sijaintitietoa säilyttäviin koordinaattiarvoihin tasaisin aikavälein jonkin vakiovektorin x- ja y-komponentit (Olettaen, että kuvaamme liikettä x, y –tasolla). Tämän liikkeen voisi ajatella olevan alkeellinen fysiikanmallinnus kappaleen liikkeestä tasolla painovoimattomassa tyhjiössä.

Renderöinti eli sovelluksen tilan esittäminen visuaalisesti riippuu ohjelmistokehyksestä oman sovelluksemme ympärillä. HTML5-tekniikalla toteutetut sovellukset mahdollistavat pelimoottorin renderöinnin toteuttamiselle muutaman vaihtoehdon, joista tärkeimmät ovat CSS muotoilujen käyttö, piirtäminen canvas komponentille ja WebGL-näytönohjainrajapinnan käyttö.

Renderöinti CSS:n avulla on varteenotettava ratkaisu pieniin selainpeleihin. CSS tarjoaa runsaasti valmiita toiminnallisuuksia visuaalisten efektien ja esimerkiksi tweenauksen toteuttamiseen. Peliobjektien yksinkertainen liikuttaminen ja asettelu voidaan toteuttaa CSS:n avulla esimerkiksi määrittämällä niiden *position* attribuutti arvoon *fixed* ja käsittelemällä *top* ja *right* attribuutteja y- ja x-koordinaattien tapaan. CSS:n avulla on kuitenkin vaikeaa aikaansaada sitä visuaalista ilmettä ja tarkkuutta, jota tavoiteltiin tässä projektissa.

Renderöinnissä canvas-komponentille voidaan käyttää perinteistä puskureihin perustuvaa tekniikkaa, jolloin piirrettävä data koostetaan ensin puskurina toimivaan tietorakenteeseen, joka lopulta piirretään näytölle ohjelmistokehykselle ominaisella tavalla. Menetelmän etu saadaan toimenpiteiden nopeudesta, sillä yksittäisen pikselin piirtäminen ruudulle on huomattavasti hitaampaa, kuin sen määrittäminen puskuriksi määritettyyn tietorakenteeseen, joka piirretään lopulta yhdellä piirtofunktiokutsulla.

WebGL mahdollistaa laitteen näytönohjaimen hyödyntämisen renderöintiprosessissa ja täten myös kolmiulotteisen kuvan esittäminen on nopeaa. Sen tuettavuus eri laitealustoilla ei kuitenkaan ole vielä täydellinen. WebGL:n käyttö tässä pelimoottorissa mahdollistaisi suuremman resoluution ja monipuolisemman valaistuksen. WebGL:n implementoiminen on yksi pelimoottorin tulevista kehityshaasteista.

2.4 Kolmannen osapuolen JavaScript pelimoottorit

JavaScriptillä on toteutettu lukuisia pelimoottoreita, jotka tarjoavat monenlaisia visuaalisia tehosteita ja valmiita rakenteita pelien tekemiseen. Tunnetuimpia näistä lienevät Unity3D:n selainversio ja kaksiulotteisiin peleihin erikoistunut Pixi.js. Tämän projektin tavoitteena oli kuitenkin toteuttaa pelimoottori, jonka avulla voi luoda dynaamisia, muokkautuvia peliympäristöjä. Tällöin ympäristön, peliobjektien ja partikkelien vuorovaikutuksessa täytyy käyttää hyvin moninaisia keinoja. Koska mikään valmis pelimoottori ei auttaisi tämän vuorovaikutuksen toteuttamisessa kovin pitkälle, oli perusteltua toteuttaa myös muut sen osat itse.

Liiallinen valmiiden kirjastojen käyttö voi johtaa ”musta laatikko” syndroomaan, joka voi olla joltain osin hyödyllistä, mutta toiselta vahingollista. Asiat saattavat tulla hoidetuksi nopeasti, mutta ilman käsitystä siitä, miten tai kuinka. Miten ja kuinka saavat todellisen merkityksensä asioiden mennessä pieleen tai suorituskyvyn noustessa ongelmaksi. (Lindley 2012, 11.) Nykyisenä teknologian nopean kehittymisen ja myös vanhenemisen aikana ohjelmistokehittäjälle on vaarallista liiaksi sitoutua tiettyyn ympäristöön. Mitä vähemmän ohjelma rakentuu tietylle ohjelmointikielelle ominaisiin piirteisiin ja kirjastoihin, sitä helpompaa sen toteuttamisen on toisella kielellä toisessa ympäristössä.

3 PELIN ESITTELY

3.1 Pelin idea ja tavoite

Pelaajan tehtävänä jokaisessa kentässä on selvitä kenttään sijoitetulle ovelle – keinolla millä hyvänsä. Mikäli ovi on kiinni, täytyy pelaajan ensin löytää kätketty avain. Avaimen ja oven saavuttaminen edellyttävät kenttäsuunnittelusta riippuen erilaisia taktiikoita. Avain saattaa esimerkiksi olla haudattuna maan alla, jolloin pelaajan tulee kaivaa tiensä sen luo. Ampiaisten pesä saattaa olla aivan oven vieressä, jolloin pelaajan tulee seurata niiden toimintaa ja ajoittaa yrityksensä tarkasti. Esteen ylittäminen saattaa edellyttää nopeasti kasvavien kasvien siementen istuttamista sen vierelle.

Pelaaja ohjaa pelihahmoaan näppäimistöllä tasohyppelypeleille ominaisella tavalla. Hahmo osaa hypätä, kiivetä sekä liikkua horisontaalisesti. Hahmo osaa myös käyttää löytämiään esineitä. Näitä toimintoja ohjataan hiirellä. Hiirellä haluttuun suuntaan klikkaamalla voidaan kaivaa maata, heittää multa- tai lumipalloja sekä kasvien siemeniä. Kontrollit on suunniteltu siten, että pelaaja käyttäisi näppäimistöä vasemmalla ja hiirtä oikealla kädellään.

Pelin tavoite on muodostua kohdeyleisölleen riittävän kiinnostavaksi, jotta nämä innostuisivat tekemään siihen itse kenttiä. Tätä varten on toteutettu kenttäeditori, joka mahdollistaa käyttäjien tekemiin kenttiin kaikki ominaisuudet, joita on käytetty pelin omien viidentoista kentän tekemisessä. Pelin kenttäeditori on pyritty tekemään mahdollisimman helppokäyttöiseksi ja sen kehitystyö jatkuu yhä.

3.2 Päävalikko

Peli aukeaa Ylen logon jälkeen alkuvalikkoon, joka on samalla myös pelin ensimmäinen kenttä. Alkuvalikko esittelee täten välittömästi pelin toiminnallisuuksia ja tarjoaa turvallisen ympäristön kokeilla hahmon liikuttamista. Alkuvalikon taustalla näkyvä kenttä on pyritty tekemään kiinnostavan näköiseksi. Sinne on asetettu muutamia peliobjekteja, jotka vuorovaikuttavat ympäristönsä kanssa, syövät kasveja ja lisääntyvät. Kuvassa 1 on nähtävillä pelin päävalikko.

Alkuvalikossa on viisi painiketta, joita käytetään klikkaamalla niitä hiirellä. “Pelaa” painikkeen kautta pääsee kentänvalinta näkymään, “Luo oma kenttä” vie kenttäeditoriin, “Syötä kentän koodi” mahdollistaa suoran siirtymisen tiettyyn kenttään ja “Ohjeet” avaa nimensä mukaisesti ohjenäkymän. ”Galaxi” painike on linkki *Galaxin* nettisivuille, joilta löytyy lisää tietoa pelistä ja koostettuja julkaisuja käyttäjien tekemistä kentistä. Ruudun vasemmassa yläkulmassa on tekstikenttä, joka kertoo kulloisenkin tilan nimen. Ruudun oikeassa alalaidassa näkyy aina pelin logo, jossa on tyylitellyillä kirjaimilla kirjoitettu pelin nimi. Logon on suunnittelusta vastasi Tamperelainen suunnittelutoimisto G360, jolta tilattiin myös pelin musiikit.

KUVA 1. Pelin päävalikko

3.3 Kentän valinta ja pelaaminen

Kentänvalinta-näkymästä käyttäjä voi valita haluamansa kentän. Seuraavan kentän avautuminen edellyttää edellisen läpäisemistä. Kenttiä voi jälkikäteen pelata uudelleen. Peli säilyttää tiedon käyttäjän vaiheesta selaimen välimuistissa, jotta peliä voi jatkaa myöhemmin samasta vaiheesta, vaikka selaimen sulkisi välissä. Tämän näkymän yhteyteen on tarkoitus julkaista myöhemmin uusia kenttäkokonaisuuksia pelaajien tekemistä kentistä. Kentänvalinta-näkymä on nähtävillä kuvassa 2.

KUVA 2. Kentänvalinnan käyttöliittymä

Peli luo kenttien ikonikuvat ottamalla kyseisen kentän bittikartasta näytteitä tarkoituksenmukaisella resoluutiolla. Näin aikaansaadaan pienennetty kuva alkuperäisestä kenttäkuvasta. Painikkeet luodaan käyttäen peliä varten toteutettua painikekäyttöliittymäkomponenttia, jolle pienennetty kenttäkuva on määritetty ikoniksi.

Tyypillinen kenttä koostuu sokkelomaisesta maastosta, staattisista ja dynaamisista kasveista, eliöistä, ovesta ja avaimesta. Kentän läpäisemisen edellytys on yksinkertainen: pelaajan tulee päästä ovelle. Mikäli ovi on lukossa, täytyy ensin etsiä avainkätkö. Suoritustapa on aina vapaa, se voi edellyttää esimerkiksi hyppimistä, kiipeämistä, kaivamista, räjäyttämistä ja rakentamista. Pelille tyypillinen kenttä on nähtävillä kuvassa 3.

KUVA 3. Tyypillinen kenttä

3.4 Kenttäeditori

Kenttäeditorin käyttöliittymä koostuu viidestä eri työkalupaletista. Kaikkia näitä paletteja voi halutessaan liikuttaa raahaamalla, jotta myös niiden taakse voi sijoittaa objekteja. Oikeaan laitaan on sijoitettu erilaisten maastotyyppien piirtotyökalut, alhaalla on peliobjektien sijoittamisen työkalu, vasemmasta yläreunasta löytyy dynaamisen maastonluonnin työkalut, niiden alta sääasetukset ja vasemmasta alakulmasta kenttäeditorin päävalikko.

Dynaaminen maastonluonti tarkoittaa sitä, että kenttään voi yhdellä painalluksella luoda tiettyjä kenttien visuaalista yhtenäisyyttä tukevia elementtejä. Tällä hetkellä toteutetut dynaamiset maastoelementit ovat: timantit, nurmikko ja köynnökset. Kuvassa 4 toteutettu kenttä on toteutettu parilla klikkauksella käyttäen dynaamisen maastonluonnin työkaluja.

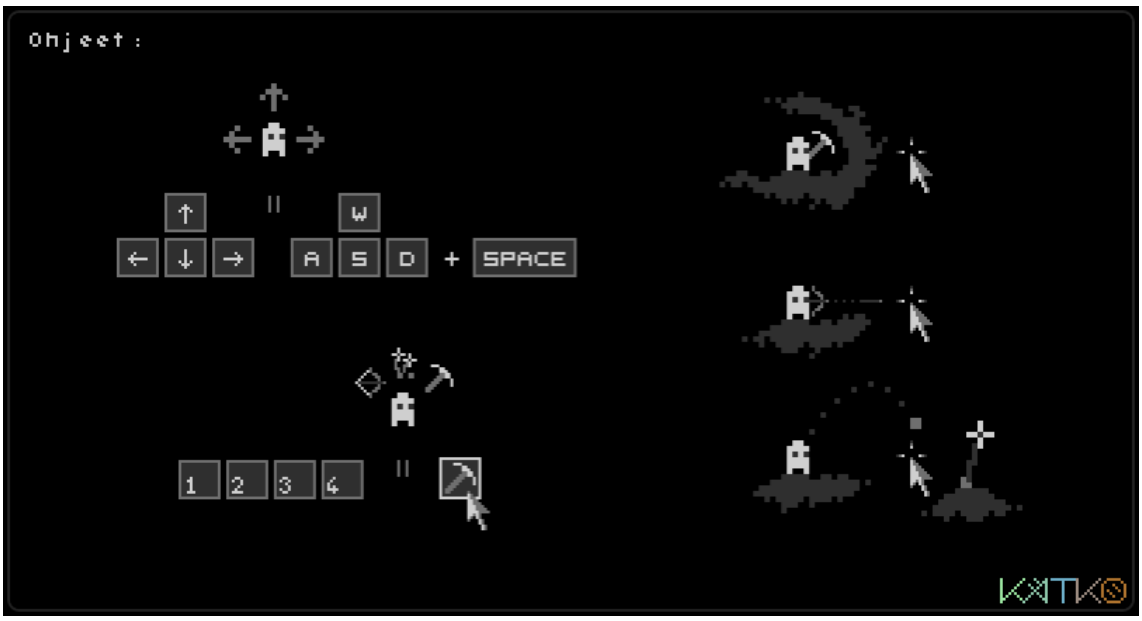
Käyttäjä voi halutessaan jakaa luomansa kentän klikkaamalla ”Jakaminen” nappulaa. Näin tehdessään käyttäjä saa linkin, jonka tämä voi jakaa esimerkiksi *Galaxin* nettisivuilla tai suoraan haluamalleen henkilölle. Myös toisen käyttäjän tekemän kentän tai valmiiden kenttien jatkojalostaminen on mahdollista, sillä käyttäjä voi halutessaan syöttää kentän koodin myös kenttäeditoriin.

KUVA 4. Kenttäeditorilla käyttäjä voi luoda omia kenttiä

3.5 Ohjeet

Pelin sisäiset ohjeet on toteutettu yhdellä infografiikkakuvalla. Siihen on pyritty sisällyttämään ohjeet hahmon liikuttamisen ja yksinkertaisimpien toimintojen kontrolleista. Kaiken muun pelissä tarvittavan tiedon oletetaan selviävän pelaajalle yrityksen ja erehdyksen kautta. Infografiikka on nähtävillä kuvassa 5.

Ohjetta testattiin käyttäjätestauksen yhteydessä esittämällä se testaajille kirjallisena. Sen todettiin olevan riittävä hahmon liikuttamisen kontrollien opettamiseen. Ohjeeseen lisättiin myöhemmin kuvassa 5 oikealla puolella näkyvät kolme osiota, jotka opastavat työkalujen ja kasvien siementen käyttöä.

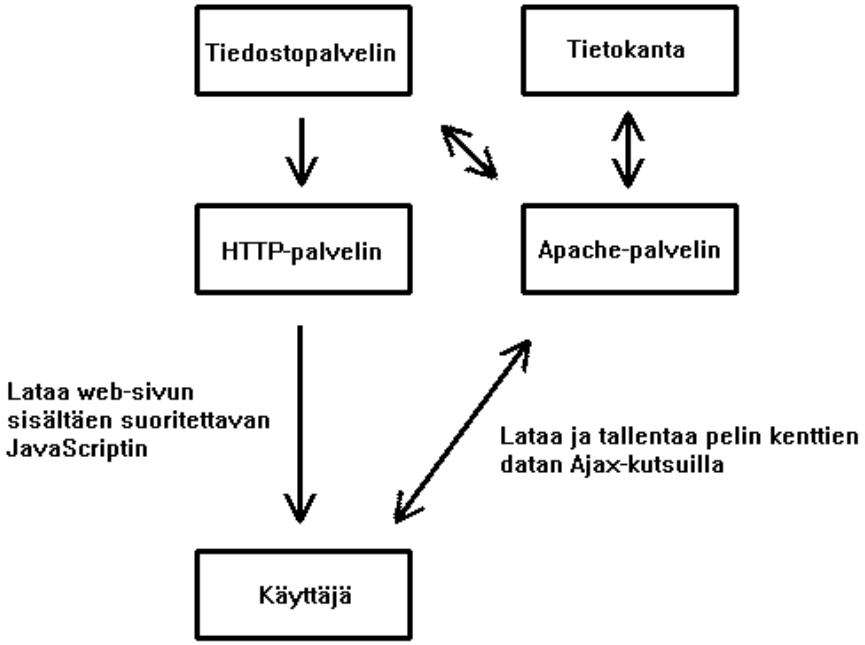


KUVA 5. Ohjenäkymä kertoo pelihahmon kontrolloinnista

4 JÄRJESTELMÄARKKITEHTUURI

Sovelluksen järjestelmäarkkitehtuuri koostuu karkeasti viidestä osasta: käyttäjästä, HTTP-palvelimesta, tiedostopalvelimesta, Apache-palvelimesta ja tietokannasta. Käyttäjät ottavat jollakin web-selaimella yhteyden HTTP-palvelimeen, joka lataa käyttäjälle pelin Web-sivun ja suoritettavan JavaScript-koodin. Peli toimii tämän koodin varassa, kunnes sen tarvitsee ladata pelin kenttiä.

Peli lataa pelattavat kentät suorituksen aikana palvelimelta. Tällöin käyttäjän Javascript ottaa yhteyden Apache-palvelimelle. Apache-palvelimelta tavoitetaan PHP:llä toteutettu ohjelma, joka tarkistaa pyynnön muodon oikeellisuuden ja etsii tämän jälkeen tietokannasta halutun kentän dataa. Mikäli kentän tiedot löytyvät tietokannasta asiakassovellukseen palautetaan linkki kentän kuvatiedostoon sekä tarvittavat lisätiedot peliobjektien sijainneista ja muista asetuksista. Pelin järjestelmäarkkitehtuurin kaavio on hahmoteltu kuvassa 6.



KUVA 6. Järjestelmäarkkitehtuuri

5 OHJELMISTOARKKITEHTUURI

5.1 Moduulipohjainen suunnittelumalli

Pelimoottori nojaa vahvasti moduulipohjaiseen suunnittelumalliin. Moduulit ovat olennainen osa minkä tahansa vankan JavaScript sovelluksen arkkitehtuuria ja ne auttavat pitämään koodin osat selkeästi eroteltuina ja järjestettyinä (Osmani 2014, 25). Moduulien implementoimiseen on useita keinoja. Pelimoottorissa käytetään *Asynchronous Module Definition* suunnittelumallia RequireJS kirjaston avulla. Kirjasto valikoitui käytettäväksi alun perin siksi, että sitä käytettiin tietyssä toimituksemme tilaamassa sisällönhallintajärjestelmässä, jonka rakenteen ymmärtämisen edellytys oli siihen tutustuminen. Tarkempi tutustuminen kirjastoon osoitti sen olevan suosittu tapa toteuttaa moduulipohjaista suunnittelumallia.

Moduulipohjainen suunnittelumalli kehitettiin alun perin keinoksi toteuttaa yksityinen ja yleinen kapselointi. Moduulien avulla kun voidaan jäljitellä luokkien toimintaa toteuttamalla objekteille yksityisiä ja yleisiä metodeita ja muuttujia. Näin vältetään julkisen nimiavaruuden saastuttamista ja estetään yhteentörmäykset toisen kehittäjän luomaan rajapintaan. (Osmani 2014, 27.)

RequireJS helpottaa uusien moduulien liittämistä koodiin poistamalla tarpeen lisätä uusien lähdekooditiedostojen viittauksia HTML-dokumenttiin. Lisäksi se mahdollistaa lähdekoodin helpon minimoimisen ja yhdistämisen yhdeksi tiedostoksi. Tämä toimenpide suoritetaan erillisellä komentorivityökalulla. HTML-dokumentissa viitataan ainoastaan RequireJS:n lähdekoodin tiedostoon yhdellä <script> -tagilla, jolle annetaan ”data-main” attribuutiksi se JavaScript tiedosto, josta ohjelman suoritus käynnistyy. Pelimoottorin lähdekoodi koostuu noin sadasta moduulista, jotka yhdistetään julkaisua varten yhdeksi tiedostoksi. Yksi moduuli saattaa käsittää yhden pelihahmon toiminnallisuudet tai vastata esimerkiksi palvelinyhteydestä. Alla on esimerkki moduulin rakenteesta. Ensimmäisellä rivillä viitataan *myFolder* alihakemistosta löytyvään *AnotherModule.js* tiedostoon. Näin se on käytettävissä moduulin sisällä. Viittausta tähän tiedostoon ei tarvitse tehdä missään muualla.

```
define(["myFolder/AnotherModule"], function(AnotherModule) {
   'use strict';
   var Module = function() {
      var self = this;
      var anotherModule = new AnotherModule();
      function initialize() {
         // called once when object is created.
      }
      this.update = function() {
         // called every frame?
      };
      initialize();
   };
   return Module;
});
```

5.2 Periyttäminen

Periyttäminen on hyödyllistä, kun sovellus tarvitsee useita eri objekteja, joilla on runsaasti samankaltaisuuksia. Ilman jonkinlaista periyttämisen toteutusta jokaiselle objektille tulisi toteuttaa samoja funktioita ja muuttujia tai rajoittua käyttämään funktiokirjastoja. Periyttämisen avulla objekteista voidaan luoda puumainen rakenne, joka välittää ylemmän tason objekteille alemman tason toiminnallisuudet. Pelimoottorissa periyttämisen hyöty ilmenee eliöitä kuvaavien peliobjektien kehittämisessä. Kaikki eliöobjektit periytyvät samasta kantaobjektista. Uuden objektin voi määrittää ylikirjoittamaan kantaobjektin toiminnallisuuksia ja ominaisuuksia tai lisätä siihen uusia.

Yleisin tapa toteuttaa periytyvyyttä JavaScriptillä, on käyttää objektien prototype ominaisuutta, mutta myös muita tekniikoita ja suunnittelumalleja käytetään paljon. On huomion arvoista, että myös kielen natiivit rakentajafunktiot perustuvat siihen. Lindley (2012, 100) suosittelee prototype periyttämiseen tutustumista, sillä sen ymmärtäminen on edellytys kielen syvemmälle ymmärtämiselle.

Eliöitä kuvaavien peliobjektien ominaisuudet on toteutettu hyödyntäen prototype-periyttämistä, erillisiä ominaisuus-moduuleita ja tietylle objektille spesifejä funktioita.

Näin on saatu pidettyä eliöiden *parent* objekti kevyenä ja saatu erotettua ominaisuudet, kuten erilaiset liikkumisen muodot tai vaikkapa lisääntymisen toiminnallisuudet selkeästi omiksi kokonaisuuksikseen.

5.3 Funktiokirjastot

Monet pelimoottorin osat tarvitsevat erilaisia yhteentörmäys- ja päällekkäisyystestejä. Suorituskyvyn ja käyttökohteen mukaan on järkevää käyttää lukuisia erilaisia menetelmiä. Useat objektit saattavat tarvita myös erilaisia laskutoimituksia kuten etäisyyksien, normaalivektorien tai lentoratojen laskentaa. Tällaisia toiminnallisuuksia voidaan toteuttaa funktiokirjastojen avulla.

Yleishyödyllisiä funktioita on koottu pelimoottorin globaalin objektin alta löytyvään ”Util” objektiin, jonka kautta niitä voi käyttää staattisen objektin kautta samaan tapaan, kuin kielestä natiivisti löytyvän ”Math” objektin funktioita. ”Util” objektista ei siis tällöin tarvitse tehdä omaa instanssiaan funktioiden käyttämistä varten. Reitinetsinnän kontrollerille on toteutettu oma ”Util” objektinsa tukemaan erilaisten reitinetsintäalgoritmien kehittämistä.

5.4 Keskitetyt kuva- ja ääniresurssit

Pelimoottori lataa kaikki tarvitsemansa kuva- ja ääniresurssit ennen käynnistymistään. Resurssien lataajalle voi määrittää takaisinkutsufunktion, jota kutsutaan latauksien valmistuttua. Resursseista voi tämän jälkeen hakea ääniä, kuvia tai kuvien tavudataa niiden nimen perusteella. Mikäli peliobjektit haluavat tehdä muutoksia ulkoasuunsa esimerkiksi satunnaistamalla väriään, ne luovat itselleen kopion alkuperäisestä bittikartasta.

Pelimoottori käyttää kaikissa ääniefektien toimenpiteissä howler.js kirjastoa. Kirjaston käyttöön päädyttiin, sillä toiminnalliset vaatimukset äänille olivat kevyet ja ne haluttiin saada toimimaan helposti. Äänimoottorin rakentaminen olisi ollut työlästä, eikä työn alkuvaiheessa ollut näkyvissä, mitä erityisominaisuuksia ääniltä edellytettäisiin, jotka vaatisivat itse tehdyn järjestelmän tuomaa kontrollia.

Kirjasto on kevyt, mutta sisältää lukuisia ominaisuuksia, joista pelimoottori käyttää nykyisessä muodossaan äänenvoimakkuuden säätöä, panerointia ja toistoa. Ääniefektien satunnaistamisen mahdollistavat toiminnallisuudet on toteutettu ”Util” funktiokirjaston kautta. Howler.js hyödyntää oletusarvoisesti Web Audio Api:a, mutta mikäli tätä ei löydy, käytetään HTML5 audiota. Mainittakoon ettei kirjaston käyttö ei ole ollut täysin ongelmatonta. Toistaiseksi tuntemattomasta syystä äänitiedostojen esilataamisen takaisinkutsufunktiot eivät tapahdu samoin kaikissa selaimissa.

6 TIETOKANNAN RAKENNE

Tietokannan tehtävä on mahdollistaa käyttäjien luomien kenttien välittäminen toisille pelaajille. Kaikki pelin kentät tallennetaan tähän tietokantaan ja ne ovat ladattavissa sieltä kaikkine tietoineen kentän koodin perusteella. Olennaisimmat tallennettavat tiedot ovat kentän objektien tiedot (mapData), kentän kuvatiedoston URI (img_path) ja kentän koodi (mapCode). Lisäksi tallennetaan tietoja kentän tekijästä ja ajankohdasta.

Kentän objektien tiedot tallennetaan tietokantaan JSON formaatissa. Tämä mapData:ksi nimetty data pitää sisällään tiedot kaikkien peliobjektien sijainneista, ominaisuuksista ja mahdollisista sääefekteistä. Peli rakentaa kunkin kentän tämän datan mukaan, mutta ei edellytä siitä löytyvän tiettyjä tietoja. MapDatasta haetaan kentän luomisen yhteydessä tietoja, mutta mikäli niitä ei löydy, käytetään oletusarvoja. Tämä mahdollistaa joustavuutta uusien ominaisuuksien kehittämisessä.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
description	varchar(10000)	NO		NULL	
created	timestamp	NO		CURRENT_TIMESTAMP	
img_path	varchar(255)	NO		NULL	
creator	varchar(255)	NO		NULL	
state	int(11)	NO		0	
mapData	varchar(10000)	YES		NULL	
mapCode	varchar(100)	YES		NULL	

KUVA 7. Tietokannan rakenne

7 PELIMOOTTORIN TOTEUTUS

7.1 Törmäystenhallinta

Peli käyttää törmäystenhallinnassa muutamia erilaisia menetelmiä riippuen siitä, mistä peliobjektista on kyse ja mihin sitä törmäytetään. Tietyt objektit edellyttävät, että niitä kuvataan tilanteesta riippuen erilaisilla muodoilla. Peliobjekti voi esimerkiksi olla maaston suhteen suorakulmio, mutta jonkin toisen objektin suhteen ympyrä, piste tai jopa bittikartta.

Eliöitä kuvaavia peliobjekteja käsitellään yleensä suorakulmioina. Törmäystenhallintaan vaikuttava alue on erotettu niiden näkyvästä koosta, jottei niiden graafisen ilmeen tarvitse olla sidoksissa törmäystenhallintaan. Peliobjektien törmäystenhallintaan vaikuttava suorakulmio on aina pelimaailman akseleiden suuntainen. Tällöin sen törmäystenhallinnan funktio toisiin suorakulmioihin nähden on alla olevan esimerkin mukaisesti kevyt ja yksinkertainen.

```
Util.hitTestObject = function(obj1, obj2) {
      if(obj1.xpos + obj1.width > obj2.xpos) {
         if(obj1.ypos + obj1.height > obj2.ypos) {
            if(obj1.xpos < obj2.xpos + obj2.width) {
               if(obj1.ypos < obj2.ypos + obj2.height) {
                  return true;
               }
            }
         }
      }
      return false;
};
```

Koska eliöitä ja pelaajaa kuvaavat peliobjektit ovat leveydeltään ja korkeudeltaan hyvin pieniä ja koska pelimaailma saattaa pitää sisällään yksittäisiä ilmassa leijuvia kuvapisteitä, peliobjektit laskevat osumatestejä pelimaailman suhteen tarkistamalla jokaisen niiden peittämän kuvapisteen. Alla esitetty funktio tarkistaa löytyykö *frameMazeData*:sta eli pelin maaston kuvapistetaulukosta tietyn peliobjektin alueelta tiettyä kuvapistetyyppiä.

```
Util.frameMazeDataAgainstBoxSpecificPixelType = function(obj, pixelType) {
      var sample;
      var endX = Math.floor(obj.x + obj.width);
      var endY = Math.floor(obj.y + obj.height);
      for(var j = Math.floor(obj.y); j < endY; j++) {
         for(var i = Math.floor(obj.x); i < endX; i++)  {
            sample = Q.frameMazeData[i + j * Q.width];
            if(sample === pixelType) {
               return true;
            }
         }
      }
      return false;
};
```

7.2 Pelimaailman rakenne ja renderöinti

Pelimaailman rakenne on suunniteltu tukemaan kahta päämäärää: toiminnallisuuksia ja renderöintiä. Kaikki rakenteelliset ratkaisut juontuvat toisen tai molempien päämäärien palvelemiseen. Ajon aikana kaikki maastodatan käsittely ja manipulointi tehdään suorituskyvyn parantamiseksi tyypitettyjen tavutaulukoiden avulla. Kuvassa 8 on kuvattu pelin keskeisten tietorakenteiden yhteyksiä. Alkuperäisestä kenttädatasta eli kentän kuvatiedostosta luodaan *BaseMazeData*:ksi nimetty Uint8ClampedArray, jonka jokainen tavu vastaa yhtä kuvapistettä. Yksi etumerkitön tavu voi täten merkitä lukuarvoa 0-255. Tämä mahdollistaa 256 maastotyyppiä erilaisiin käyttötarkoituksiin. Pelimoottorin asetuksissa on määritetty tietyt vakioidut lukuarvot vastaamaan tiettyjä pelimaailman elementtejä. Lukuarvo 0 merkitsee tyhjää, 1 kaivettavaa maata, 3 kasviperäistä kuvapistettä, 6 kiveä, jonka päälle on satanut lunta jne.

BaseMazeDatan rinnalla ylläpidetään toista tavutaulukkoa *ColorMazeDataa*, joka säilyttää kunkin pikselin väriarvon. Värit tallennetaan 32 bittisinä RGBA muodossa, joten kunkin pikselin väriarvon tallentamiseen käytetään neljä tavua. Nämä kaksi taulukkoa toimivat renderöinnin ja toiminnallisuuksien lähtökohtana.

Pelimoottori tukee monipuolista partikkelijärjestelmää. Partikkelit voivat valaista ympäristöään, muuttaa väriään, omata tietyn painovoiman, käyttäytyä kiinteän aineen tai nes-

teen tavoin ja vuorovaikuttaa muiden peliobjektien kanssa. Partikkelien valaistuksen ja vuorovaikutuksen itsensä sekä muiden objektien kanssa voisi tehdä *BaseMazeData*:n kautta. Tällöin partikkelien kuitenkin täytyisi muistaa ylikirjoittamansa *BaseMazeData*:n arvo ja rajoittua liikkumaan pelkästään esimerkiksi tyhjän arvon omaavalla alueella. Ratkaisuna *BaseMazeData*:sta luodaan joka ohjelmasyklissä kopio, joka on nimetty *FrameMazeDataksi*. Tähän taulukkoon partikkelit voivat piirtää itsensä ja tämän taulukon pohjalta myös peliobjektit tekevät pelimaailman törmäystenhallintansa.

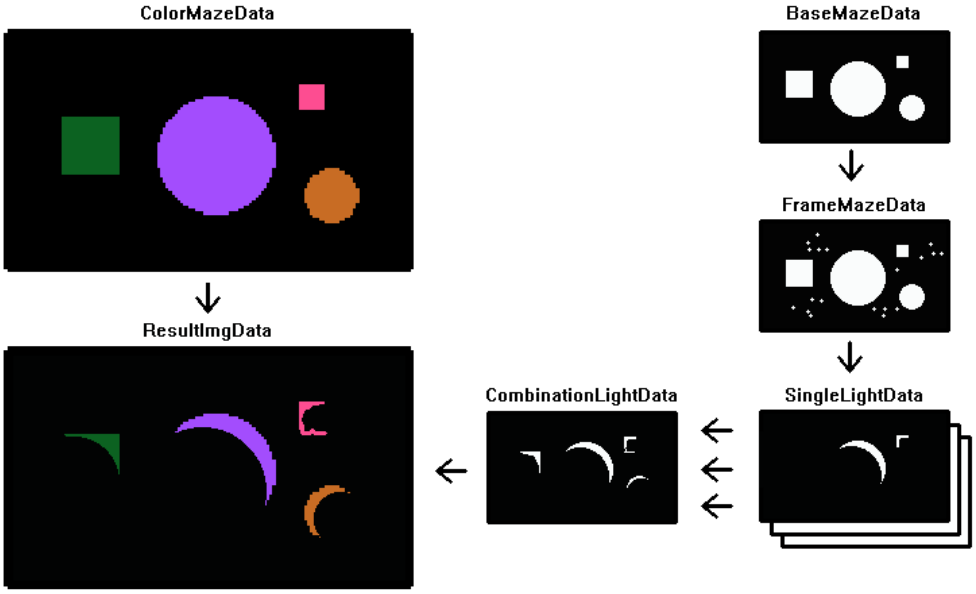
Mikäli moottori ei käyttäisi erillistä valaisua, voitaisiin peli renderöidä piirtämällä canvakselle ensin *ColorMazeData* ja sitten peliobjektit tämän päälle. Tämä kuitenkin aiheuttaisi tarpeettoman monta suhteellisen raskasta canvas-operaatiota eikä dynaamista valaistusta olisi helppoa toteuttaa. Lukuisilta canvas operaatioilta vältytään piirtämällä peliobjektit ensin *ResultImgData*:ksi nimettyyn tavutaulukkoon, jonka alpha-kanava on kaikkien valoelementtien yhdistelmä *CombinationLightData*. Tällä tavoin pelimoottori voi renderöidä näkymän yhdellä ainoalla canvas:n `putImageData` funktiokutsulla.

Tyypitettyjen taulukoiden indeksointi on nopeaa, sillä niiden sisältämä data sijaitsee tietokoneen muistissa peräkkäisissä osoitteissa. Aikaisemmin mainitut taulukot kuvaavat kaikki kaksiulotteista dataa, mutta niiden tieto on tallennettu tavalliseen yksiulotteiseen taulukkoon. Oikean kuvapisteen manipulointi kaksiulotteisten koordinaattien perusteella edellyttää tietoa kaksiulotteisen taulukon leveydestä. Tietyn kuvapisteen x, y sinisen kanavan indeksi voitaisiin löytää alla näkyvällä kaavalla, jossa luku 4 juontuu kuvadatan RGBA rakenteesta ja luku 2 sinisen kanavan sijainnista siinä.

```
indeksi = x * 4 + y * 4 * (kuvan leveys) + 2
```

Tällä tavoin laskettua indeksiä voitaisiin käyttää esimerkiksi muuttamaan tietyn kuvapisteen sinisen kanavan arvoa pelin maastossa. Maaston tietorakenteisiin pääsee käsiksi mistä tahansa objektista globaalin muuttujan *Q* kautta, joka sisältää viittaukset pelin keskeisiin tietorakenteisiin ja asetuksiin. Kuvapisteen x = 100, y = 200 vihreälle kanavalle voisi asettaa arvon 123 alla esitetyllä komennolla.

```
Q.ColorMazeData[100 * 4 + 200 * 4 * Q.width + 1] = 123;
```



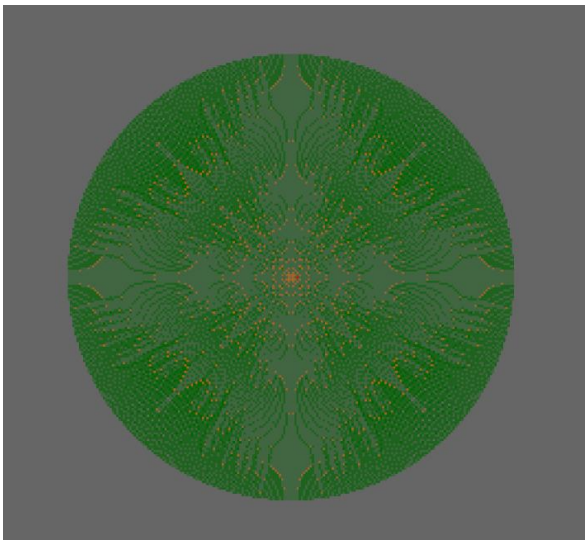
KUVA 8. Kaavio renderöinnistä ja pelimaailman keskeisistä tietorakenteista

7.3 Valaistus

Peli hyödyntää kahta erilaista tapaa luoda valoja. Toinen tapa on suorituskyvyn kannalta kevyt, mutta ei luo varjoja. Siinä yksittäisten kuvapisteiden valoarvoa kasvatetaan lisäämällä niihin arvo ennalta lasketusta oikean kokoisesta keskeltä reunoille himmenevästä gradienttiympyrästä. Valo ei tällöin huomioi ympäristöään vaan valaisee myös mahdollisten seinämien lävitse. Tätä ei kuitenkaan pienten valojen kohdalla juurikaan huomaa ja tekniikan ollessa moninkertaisesti säteenseurantaa nopeampaa, yksinkertaisten valojen käyttö on useissa kohteissa järkevää. Kuvassa 9 keltainen kukka on valaistu yksinkertaisella valolla.

KUVA 9. Pelimoottori tukee sekä säteenseurantaa, että yksinkertaista valoa. Tietyt peliobjektit reagoivat valon määrään.

Raskaampi, mutta näyttävämpi tapa laskea kuvapisteiden valoarvoja on lähettää valon alkupisteestä valonsäteitä ja asettaa säteen kantama valoarvo pienenemään, mikäli se kohtaa seinämiä. Valonsäteiden normaalivektorit on tällöin syytä laskea etukäteen ja muutoinkin pyrkiä tiivistämään sekä yksinkertaistamaan algoritmin useimmin toistuvia osia. Säteenseurantaa on pyritty optimoimaan selvittämällä etukäteen pisteet, joissa uusi valonsäde ensimmäisen kerran osuu ennalta valaisemattomaan kuvapisteeseen. Hyöty saadaan siitä, ettei samoja kuvapisteitä valaista useaan kertaan. Kuvassa 10 on esitetty symmetrisesti luotujen valonsäteiden alkupisteet keltaisella pisteellä. Kuvassa 9 hiiren kursorista, pelaajan hahmosta ja ovesta lähtee säteenseurannalla toteutettua valoa.



KUVA 10. Symmetrisesti luotujen säteiden alkupistekartta

7.4 Käyttöliittymäkomponentit

7.4.1 Tekstikenttä ja fontti

Sovellusta varten toteutettiin oma fontti. Koska pelin resoluutio on hyvin pieni, (400 x 212) myös fontin kirjainmerkkien tulee olla mahdollisimman pieniä. Muutoin tekstikentät yksinkertaisesti täyttäisivät liian suuren alueen pelin näkymästä. Pienin fontti voitaisiin toteuttaa, mikäli ei tehtäisi eroa isojen ja pienten kirjainmerkkien välillä. Tällaisen fontin luettavuus olisi kuitenkin heikko, eikä pienistä kirjainmerkeistä haluttu luopua. Olemassa olevia pikselifontteja tutkimalla päädyttiin lopulta fonttiin, jonka yksittäisen kirjainmerkin koko on 5 x 7 kuvapistettä.

Kyseinen koko on jotakuinkin pienin mahdollinen, jolla voidaan toteuttaa fontti sekä isoilla, että pienillä kirjainmerkeillä, siten ettei tekstin luettavuus suuresti kärsi. Fontti on suunniteltu käyttämään tasakokoista kirjainväliä. Pienessä pikselifontissa toteutuksen vaihtoehdot tietyille kirjainmerkeille ovat toisinaan vähäiset ja fontti pohjautuu valmiisiin ilmaisiin pikselifontteihin. Fontti on nähtävillä kuviossa 11.

KUVA 11. Peliä varten toteutettu mahdollisimman pienikokoinen fontti

Oman tekstikentän ja fontin tekemiseen ryhdyttiin, sillä sen toiminnalliset vaatimukset eivät olleet aluksi kovin suuret. Tavallisesti valmiiden käyttöliittymäkomponenttikirjastojen tekstikentät ovat toiminnoiltaan varsin monipuolisia. Ne pystyvät pelkän tekstin esittämisen lisäksi yleensä myös vastaanottamaan käyttäjän syötteitä, sallivat tekstin maalaamisen hiirellä, värin vaihtamisen, fontin vaihtamisen, fontin koon muuttamisen, rivittämisen ja paljon muuta. Vaikka tekstikentälle myöhemmin toteutettiin joitain näistä toiminnoista, sen alkuperäiset vaatimukset eivät olleet järin laajat.

-Tekstikentän sijaintia tulee voida muuttaa.
-Sen tekstiä tulee voida muuttaa.
-Sille tulee voida asettaa kuuntelijafunktio klikkaukselle.

-Sen väriä tulee voida muuttaa.
-Sen tulee olla kevyt.

Yksinkertaistetusti ilmaistuna tekstikenttä on objekti, jolle voidaan määrittää haluttu teksti merkkijonona, jonka perusteella se rakentaa oman kuvadatansa kirjainmerkki kerrallaan kopioimalla fontin sisältävää kuvadataa oikeista paikoista. Tämä kuvadata piirretään lopulta oikeaan paikkaan ruudulla tekstikentän koordinaattien perusteella. Ensimmäinen versio tekstikentästä oli hidas, sillä se kopioi kuvadatansa uudelleen jokaisessa ohjelmasyklissä, vaikka sen päivittäminen on tarpeen vain tekstin muuttuessa. Kun tämä virhe korjattiin, tekstikenttä osoittautui erittäin käyttökelpoiseksi.

Jotta tekstikenttää saatettiin käyttää painikkeena, sille toteutettiin yksinkertainen tapahtumakuuntelijajärjestelmä. Erilaisille hiiren tapahtumille (”event”) voidaan asettaa kullekin oma takaisinkutsufunktionsa, jonka olemassaolo aktivoi tapahtumien jokaisessa ohjelmasyklissä tapahtuvan testaamisen. Testaamisessa käytetään samoja törmäystenhallinnan funktioita, kuin peliobjekteissa. Alla on esimerkit tekstikentän julkisesta funktiosta, jolla takaisinkutsufunktioita rekisteröidään ja yksityisestä funktiosta, joka tarkistaa onko tekstikenttää klikattu.

```
this.addEventListener = function(type, callback) {
   switch(type) {
      case "click":
         self.clickHandler = callback;
         break;
      case "hover":
         self.hoverHandler = callback;
         break;
      case "hoverEnter":
         self.hoverEnterHandler = callback;
         break;
      case "hoverLeave":
         self.hoverLeaveHandler = callback;
         break;
   }
};
```

```
function checkAndActForClick() {
   if(Q.click) {
      if(Q.Util.hitTestPoint(Q.mouseX, Q.mouseY, self)) {
         buttonsound.play();
         self.clickHandler(self);
      }
   }
};
```

7.4.2 Painike

Vaikka klikattava tekstikenttä riitti aluksi painonapiksi, kenttäeditorin vaatimukset edellyttivät pidemmälle vietyä painikekomponenttia. Painike koostuu kehyksestä ja taustaväristä sekä haluttaessa tekstikentästä ja ikonikuvasta. Painike on parametrisoitavissa pohjaan jääväksi (“toggleable”) tai tavalliseksi automaattisesti palautuvaksi (“normal”). Painikkeita käytetään sovelluksen valikoissa, pelaajan hahmon käyttöesineen valinnassa ja kenttäeditorin työkalujen valinnassa.

Kenttäeditoria varten toteutettiin mahdollisuus raahata käyttöliittymän osia, jotta myös niiden taakse voisi sijoittaa peliobjekteja ja piirtää maastoa. Painikkeet voivat tällöin sijoittua toistensa päälle ja ovat joka tapauksessa aina myös piirtokelpoisen alueen päällä. Tämä aiheuttaa helposti ongelmia, sillä on harvoin tarkoituksenmukaista, että yhdellä klikkauksella vaikutetaan useaan toimintoon. Järjestelmä vastaa ongelmaan kuuntelemalla käyttäjän syötteitä keskitetysti ja tarjoamalla niitä tarvitseville objekteille rajapinnan, joka mahdollistaa syötelippujen resetoinnin sekä hiiren painallusten alkuperän määrittämisen. Käytännön tarve näille ominaisuuksille nousee esimerkiksi piirtotyökalun toiminnasta. Piirtäminen toteutuu, mikäli hiiren painike on pohjassa, mutta se ei saa toteutua, mikäli hiiren kursori oli painettaessa jonkin painikkeen päällä.

7.4.3 Palette

Palette:ksi nimetty käyttöliittymäkomponentti yhdistää tekstikenttiä ja painonappeja ryhmiksi. Sen nimi tulee sen alkuperäisestä käyttötarkoituksesta toimia kenttäeditorin väripalettina. Osuvampi nimi voisi olla ”Panel”. Se huolehtii saamiensa parametrien mukaisesti komponenttien asettelusta joko vertikaalisesti, tai horisontaalisesti ja mahdollistaa tarvittaessa kokonaisuuden raahaamisen hiirellä. Palette:lle voi myös määrittää taustavärin ja reunuksen sekä näiden läpinäkyvyyden.

Palette tarjoaa myös mahdollisuuden asettaa yhdellä käskyllä useiden komponenttien tiloja tai poistaa niitä näkyvistä. Palettea käytetään kaikkien pelin käyttöliittymäkomponenttien hallinnassa, mikäli komponentteja on yhdessä ryhmässä enemmän kuin yksi. Komponentin voi sijoittaa myös toisen samanlaisen sisään, jolloin voidaan luoda monimutkaisempia asetteluja. Ominaisuutta käytetään esimerkiksi kenttäeditorin ohjeissa.

7.5 Reitinetsintä

7.5.1 Algoritmin tausta

Reitinetsinnän algoritmeilla pyritään selvittämään optimaalisin reitti kahden pisteen välillä tietyssä ympäristössä. Reitinetsinnän käyttökohteita ovat mm. tietoliikenteen reititys, robottien ohjaaminen, asevoimien simulaatiot ja pelit (Yap 2005). Tekniikan merkittävän hyödyntämisen peleissä voi katsoa alkaneen *Westwood Studios*:n vuonna 1992 julkaisemasta *Dune II* pelistä, jolloin sen mahdollisuudet loivat pohjan suuresti menestyneelle reaaliaikastrategian peligenrelle. Reitinetsintä vapautti tällöin pelaajan kontrolloimasta suoraan tietyn peliobjektin jokaista liikettä ja mahdollisti pelit, joissa pelaaja kontrolloi yhtäaikaisesti suurta määrää peliobjekteja.

Reitinetsinnän hyödyntäminen peleissä voi tarjota mahdollisuuksia myös nykypäivän haasteisiin. Yksi mobiilipelien haasteista on peliobjektien kontrollointi. Kosketusnäyttö on pelikontrollerina nykymuodossaan huomattavasti oikeita painikkeita tai hiirtä epätarkempi ja vaikeakäyttöisempi. Lisäksi pelaajan omat sormet muodostuvat helposti

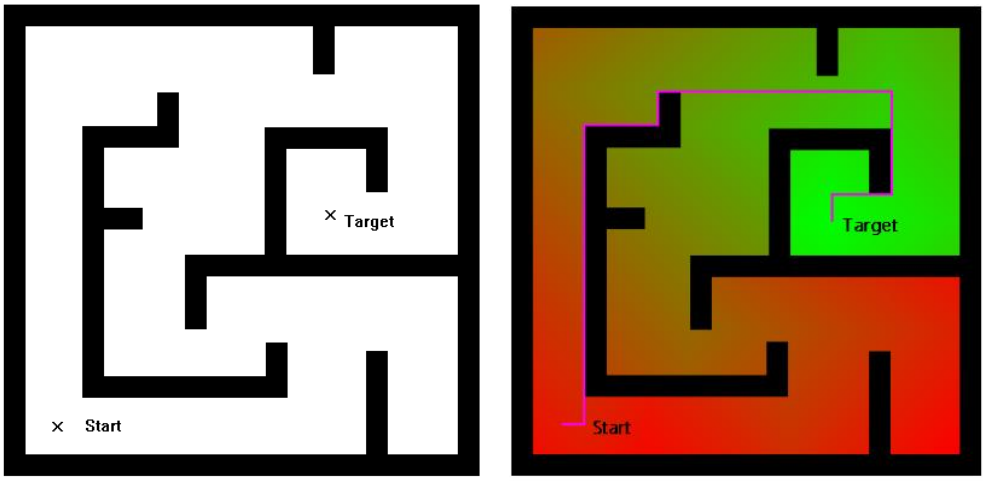
näköesteeksi. Reitinetsinnän hyödyntäminen voi vastata näihin ongelmiin vähentämällä käyttäjän peliobjekteille antamien komentojen määrää.

Reitinetsinnän algoritmit pohjautuvat usein Alankomaalaisen tietojenkäsittelytieteilijän Edsger W. Dijkstran vuonna 1956 kehittämään algoritmiin, joka on nimetty hänen mukaansa. Seuraava esimerkki kuvaa, kuinka reitinetsintäalgoritmeja Dijkstran algoritmilla yksinkertaisimmin toteutetaan. Tämän jälkeen käsitellään peliä varten valittua algoritmia ja sen sopivuutta käyttökohteeseen.

7.5.2 Esimerkki algoritmin toiminnasta

Esimerkki kuvaa reitinetsintää kaksiulotteisessa ympäristössä, jonka alkiot ovat neliön muotoisia ja jokaisella alkiolla on täten neljä ympäröivää alkiota. Diagonaalista liikettä ei sallita. Kirjainmerkki ”X” kuvaa seinämää, ”S” aloituspistettä ja ”O” kohdetta. Algoritmi on Dijkstran algoritmin variaatio, joka selvittää lyhimmän mahdollisen reitin sokkelon läpi. Kuvat 6 ja 7 kuvaavat esimerkkialgoritmin toimintaa.

Algoritmi aloittaa työnsä lähtöpisteen sijaan kohteesta. Tämä johtuu algoritmin toimintatavasta. Algoritmin ensimmäinen vaihe ei vielä palauta tiettyä reittiä kahden pisteen välillä, vaan kaksiulotteisen taulukon johon on tallennettu jokaisen saavutettavissa olevan sijainnin todellinen etäisyys kohteesta. Tietty reitti lähtöpisteen ja kohteen välillä selvitetään vasta tämän taulukon avulla. Näin luodusta taulukosta voidaan selvittää reitti kohteeseen mistä pisteestä tahansa, eikä aloituspisteellä ole täten vielä merkitystä. Aloituspisteestä voi kuitenkin olla hyötyä, mikäli haluamme laskentaresurssien säästämiseksi keskeyttää algoritmin, sen tavoitettua kyseisen pisteen. Kuvassa 12 esitetään yllä kuvailtu kaksiulotteinen taulukko siten, että kunkin kuvapisteen etäisyys kohteesta on suhteutettu väriliukuun vihreästä punaiseen. Mitä punaisempi kuvapiste on, sitä kauempana se on kohteesta.



KUVA 12. Reitinetsintäohjelmalla ratkaistu sokkelo

7.5.3 Pseudokoodi reitinetsinnästä

Alustus

Luo lista koordinaatteja varten. Kutsutaan listaa jonoksi. Lisää siihen loppupisteen koordinaatti, jolle on lisätty laskurimuuttuja, jonka arvo on 0.

Toistorakenne

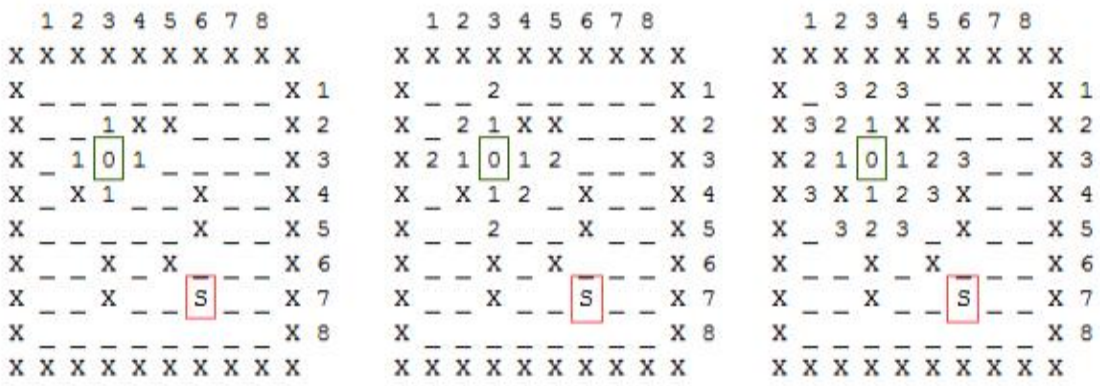
Jokaiselle jonon koordinaatille, tee seuraava:

1. Luo lista neljästä ympäröivästä koordinaatista, joiden laskurimuuttuja on alkuperäisen koordinaatin laskurimuuttuja + 1.

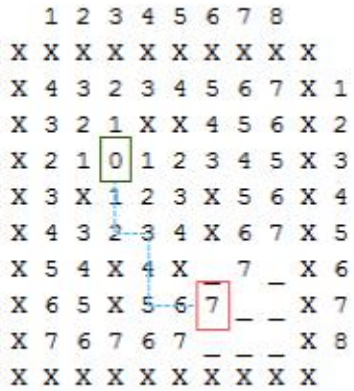
2. Tarkista nämä neljä koordinaattia seuraavilla ehdoilla:

1. Jos koordinaatissa on seinää, poista se listasta.
2. Jos jonossa on elementti, jolla on sama koordinaatti ja sama tai korkeampi laskurimuuttuja, poista se listasta.
3. Lisää jäljelle jääneet koordinaatit jonoon.
4. Etene jonon seuraavaan koordinaattiin.

Kuvassa 13 on esitetty esimerkin algoritmin toiminnan kolme ensimmäistä toistoa. Algoritmi saavuttaa kohteensa seitsemännellä toistolla, jonka jälkeen lyhin mahdollinen reitti aloituspisteestä kohteeseen on löydettävissä aloittamalla lähtöpisteestä ja kulkemalla ympäröivistä sijainneista aina siihen sijaintiin, jonka laskurimuuttujan arvo on pienin. Tämä reitti on esitetty kuvassa 14 sinisellä katkoviivalla.



KUVA 13. Reitinetsinnän kolme ensimmäistä toistoa



KUVA 14. Ratkaisu löydettiin seitsemännellä toistolla

7.5.4 Toteutukseen valittu algoritmi

Dijkstran algoritmi löytää aina nopeimman reitin, mutta sovelluksesta riippuen tämä ei aina ole välttämätöntä. Täysi varmuus lyhimmästä mahdollisesta reitistä vaatii paljon laskentaa. Dijkstran algoritmin tavallisin sovellus etsii reittiä joka puolelta leviten ympäristöönsä kuin tulva. Se etsii reittiä yhtä innokkaasti päinvastaisesta, kuin kohteen suunnasta.

Kompromissiratkaisua voi hakea algoritmin tunnetuista muunnoksista. Niille on ominaista laskea “heuristiseen arvoon” perustuva järjestys, joka toimii painotuksena sille, mikä alkio kannattaa selvittää seuraavaksi. Täten kaikkia mahdollisia reittejä ei välttämättä selvitetä vaan etsitään ratkaisua ensin heuristisen arvon painottamista alkioista. Tämä arvo voi olla yksinkertaisesti tutkittavan alkion etäisyys alkupisteestä, jolloin algoritmi etsii ensin reitin joka kulkee alkupistettä kohti. Ylimääräisen arvon laskeminen ja vertailu kuluttaa laskentaresursseja, mutta on etenkin suuren avoimen tilan sokkeloissa huomattavasti alkuperäistä tekniikkaa nopeampaa.

Kuvasta 15 ilmenee, kuinka käyttökelpoinen reitti on löydetty, ilman että kaikkia eri reittivaihtoehtoja on selvitetty. Algoritmi on aloittanut työnsä kuvan oikean laidan ampiaispesästä ja edennyt painottaen alkioita, jotka ovat alkupisteen eli ampiaisen suunnassa. Keskivaiheilla on tutkittu mahdollisuutta löytää reitti esteiden yläpuolelta, mutta etäisyysarvojen tarkastelu on ohjannut algoritmin pian oikeaan suuntaan.

KUVA 15. Reitinetsinnän debug-näkymä

7.6 Reitinseuranta ja ballististen lentoratojen määrittäminen

Ballistinen lentorata on reitti, jota kappale kulkee, kun se pudotetaan, heitetään, laukaistaan tai ammutaan, mutta se ei saa lentonsa aikana lisää energiaa. Kappaleen Lentoradan määrittää tällöin täysin sen lähtönopeus ja suunta yhdessä siihen kohdistuvan gravitaation ja mahdollisen ilmanvastuksen kanssa (de Carpentier 2014, 1). Tässä pelitoteutuksessa ei oteta huomioon ilmanvastusta, joten kappaleiden lentoratojen laskeminen hie-

man yksinkertaistuu. Koska ilmanvastusta ei oteta huomioon, voidaan puhua tyhjiölentoradoista.

Lentoratojen laskemista käytetään pelissä kun peliobjektien halutaan heittävän jotakin tai itse hyppäävänsä tiettyyn pisteeseen. Tällöin ohjelma laskee kulman ja lähtönopeuden, jotka objektille määritettäessä vievät sen kohteeseen. Toisinaan ohjelma myös selvittää etukäteen onko kyseinen lentorata vapaa esteistä objektin kuljettavaksi. Täten voidaan taata heiton tai hypyn onnistuminen jokaisella käyttäjälle näkyvällä yrityksellä, vaikka maasto sisältäisikin esteitä.

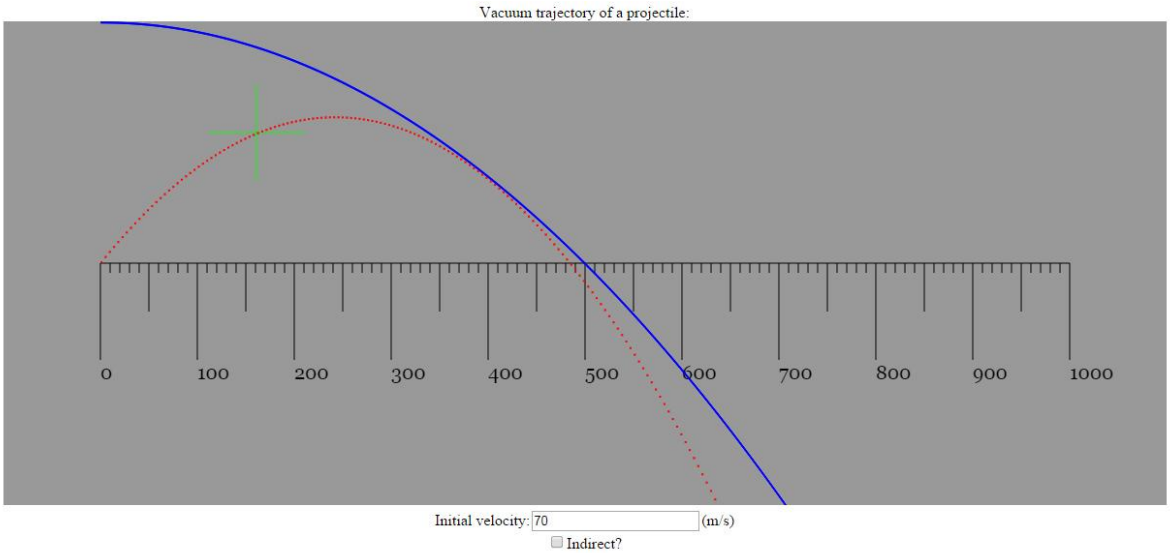
Poikkeuksellisin käyttö lentoradoille on ampiaisten reitinseurannassa käytetty sovellus. Vaikka luvun alussa sanottiinkin ballististen lentoratojen perustuvan siihen, ettei kappale saa lentonsa aikana lisää energiaa, tässä sovelluksessa kappale saa lisää energiaa ja ampiaisen lentämisen sovellus koostuukin suuresta määrästä hyvin lyhyitä ballistisia lentoratoja.

Objektin kyky selvittää reitti on hyvä lähtökohta mielenkiintoisemmille toiminnallisuuksille, mutta on vielä hieman irrallaan sovelluksen todellisuudesta. Jos objektin sallitaan liikkuvan reittiä suoraviivaisesti, se jättää silloin noudattamatta pelimaailman fysikaaliset lait. Kiinnostavampaa onkin, miten objektit saadaan noudattamaan pelin fysiikan sääntöjä eli gravitaatiota ja törmäystenhallintaa, mutta silti seuraamaan reittiään.

Sivulta päin kuvatussa pelissä maan pinnalla kulkevien objektien suhteen ongelma on suhteellisen haastava. Haastetta lisää tämän pelitoteutuksen maasto, joka on täysin satunnaista. Ilmassa lentävien objektien suhteen tehtävä on hieman helpompi. Lentäville objekteille on sallittua “tehdä siiveniskuja” eli suunnata ennalta määritetyissä rajoissa oleva voima haluamaansa suuntaan. Käytännössä reitinetsintää käyttävät lentävät objektit määrittävät tietyin väliajoin tämän voiman ja sen suunnan noudattamaan haluamaansa kohteeseen tilaamaansa reittidataa.

Objektit etsivät tällöin tietyltä pienehköltä etäisyydeltä itsestään reittidatan pistettä, joka on kaikkein arvokkain eli sisältää pienimmän etäisyyden kohteeseen. Tämän jälkeen objekti suhteuttaa siiveniskun voimakkuuden omaan etäisyyteensä valittuun pisteeseen ja laskee siiveniskun suuntauksen kappaleen tyhjiölentoradan määrittävällä kaavalla. Seurauksena on hyönteisen lentämistä muistuttava ilmassa hyppelehtivä liike. Kuvassa

16 esitetty on käyttöliittymänäkymä sovelluksesta jolla lentoratojen määrittämistä testattiin. Vihreä risti on käyttäjän määrittämä kohdepiste, sininen paraabeli kuvaa tietyllä lähtönopeudella saavutettavien etäisimpien pisteiden joukkoa ja punainen paraabeli kuvaa lentorataa, joka kulkee asetetun kohdepisteen kautta.



KUVA 16. Sovellus tyhjiölentoratojen määrittämiselle

Nopein mahdollinen reitti pelin maastossa satunnaisten pisteiden välillä kulkee lähes poikkeuksetta jossain kohdassa aivan seinämän vieressä. Tämä aiheuttaa juuttumisen ongelmia, mikäli peliobjektin yhteentörmäyksiä laskeva on suuri. Primitiivinen ratkaisu on kieltää reitinetsintää ikinä kulkemasta liian läheltä seinää, elävämpi ratkaisu on tehdä seinän läheisyydestä reitinetsintäalgoritmille “epämieluisaa”. Dijkstran algoritmin muunnelma, joka käyttää parhaan seuraavan alkion valinnassa “Heuristista arvoa” soveltuu hyvin ominaisuudelle, joka tekee tietyistä sijainneista epähalutumpia. Sokkeloa kuvaavaan taulukkoon lasketaan tällöin etukäteen lukuarvot, jotka tekevät alueesta seinän lähellä lineaarisesti epäsuotuisamman, mutta kuitenkin mahdollisen kulkualueen. Tekniikkaa on esitelty kuvassa 15, jossa alueen epäsuotuisuutta kuvaa punaisen sävyn voimakkuus.

Koska pelin maasto saattaa muuttua pelin aikana, on reitinetsinnän nopeuttamiseksi pidettävä yllä jokseenkin ajantasaista reittikarttaa, johon on valmiiksi laskettu epämieluisan alueen arvot. Tätä karttaa olisi raskasta päivittää kokonaan jokaisessa syklissä. On järkevämpää vakioida sen päivittäminen tiettyyn määrään syklejä. Mikäli pelaaja

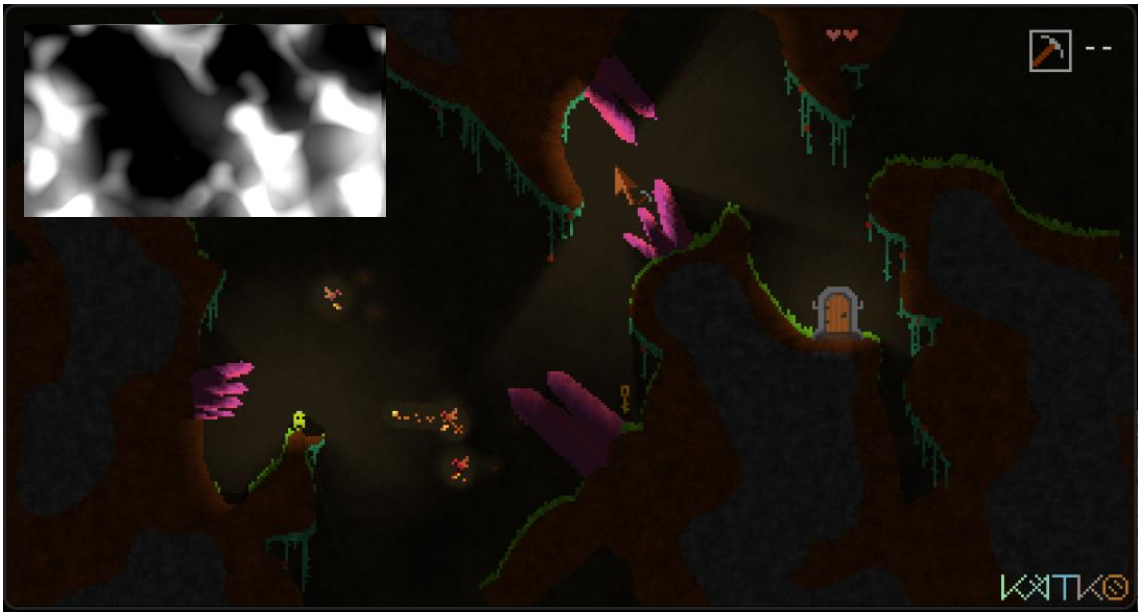
tekee maastoon muutoksia esimerkiksi kaivamalla, reitinetsintää käyttävät peliobjektit huomaavat muutokset reittikartan päivittymisen vaiheesta riippuen muutaman sekunnin viiveellä.

7.7 Perlinin kohina

7.7.1 Alkuperä ja käyttökohteet

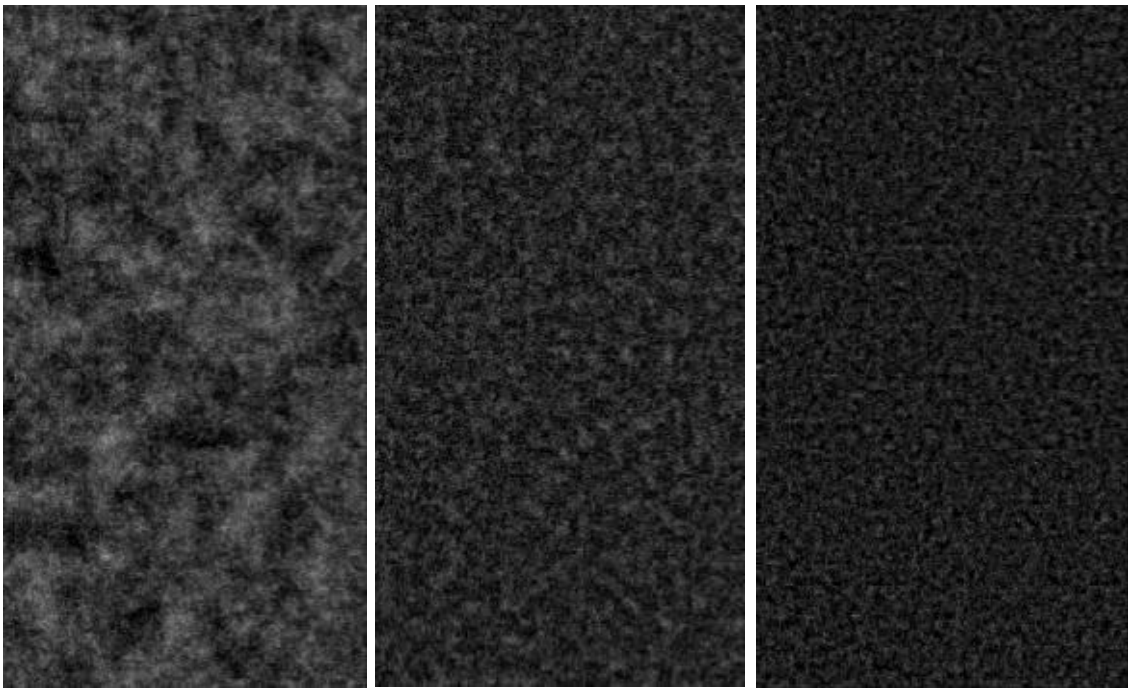
Tietty määrä satunnaisuutta voi olla hyvä asia ohjelmoitaessa orgaanisia, elämää jäljitteleviä toiminnallisuuksia. Ainoana johtavana prinsiippinä satunnaisuus ei välttämättä kuitenkaan ole luonnollista. Algoritmi, joka tunnetaan kehittäjänsä Ken Perlinin mukaan ottaa tämän konseptin huomioon. (Shiffman 2012, 17.) Yhdysvaltalainen tietokonetieteilijä Ken Perlin kehitti 1980-luvulla algoritmin, jolla voi tuottaa ohjelmallisesti monenlaisia visuaalisia efektejä, tekstuureita ja kolmiulotteisia muotoja. Algoritmi tuottaa satunnaista, mutta oikealla tapaa kerrostettuna ja parametrisoituna luonnolliselta rakenteelta näyttävää kohinaa halutuissa ulottuvuuksissa. Menetelmää käytetään usein luonnon ominaisuuksien simuloinnissa ja proseduraalisesti luodussa grafiikassa. Suuren kuvan luominen vie suhteellisen paljon laskentatehoja, mikä kannattaa ottaa huomioon sovelluksessa.

Kohinaa käytetään pelissä kahteen tarkoitukseen. Ensimmäinen käyttökohde on kenttäeditorissa. Pelaajan painaessa “Satunnaista” painiketta, sovellus luo kuvadatan Perlinin kohinan fraktaalikohinaksi kutsuttua sovellusta, joka muodostaa pelin kentän maaston. Fraktaalikohinassa asetetaan päällekkäin eri resoluutiolla, satunnaisuudella ja voimakkuudella luotuja kohinakuvia. Tämä luo vaikutelman luonnollisesta rakenteesta, jossa yksityiskohtien koko on suhteellinen niiden voimakkuuteen. Parametreja muokkaamalla valittiin kenttäeditoriin fraktaalikohina, joka muodostaa kenttäsuunnitteluun sopivia muotoja. Kuvassa 17 näkyy kuvakaappaus kentästä, jossa on käytetty satunnaista maaston generointia. Kuvan vasemmassa yläkulmassa näkyy alkuperäinen algoritmilla tuotettu kuva, jonka pohjalta kentän maasto on luotu.



KUVA 17. Perlinin kohinalla satunnaistettu maastorakenne ja lähdekuva

Toinen kohinan käyttökohde on myös usein peleissä käytetty. Sopivalla fraktaalikohinalla luodaan eri maastotyypeille omat tekstuurinsa. Nämä bittikartat tarvitsevat paljon pienempiä yksityiskohtia, mutta ovat muuten jokseenkin samanlaisia kuin maaston satunnaistamiseen käytetty kohina. Tekstuurit eivät tosin perinteisten tekstuurien tapaan vaikuta kuvapisteiden väriin, vaan kuvapisteiden läpinäkyvyys-arvoon luoden mielenkiintoisen vaikutelman yhdessä liikkuvien valojen kanssa. Erilaisia pelissä käytettyjä valotekstuureita on esitetty kuvassa 18.

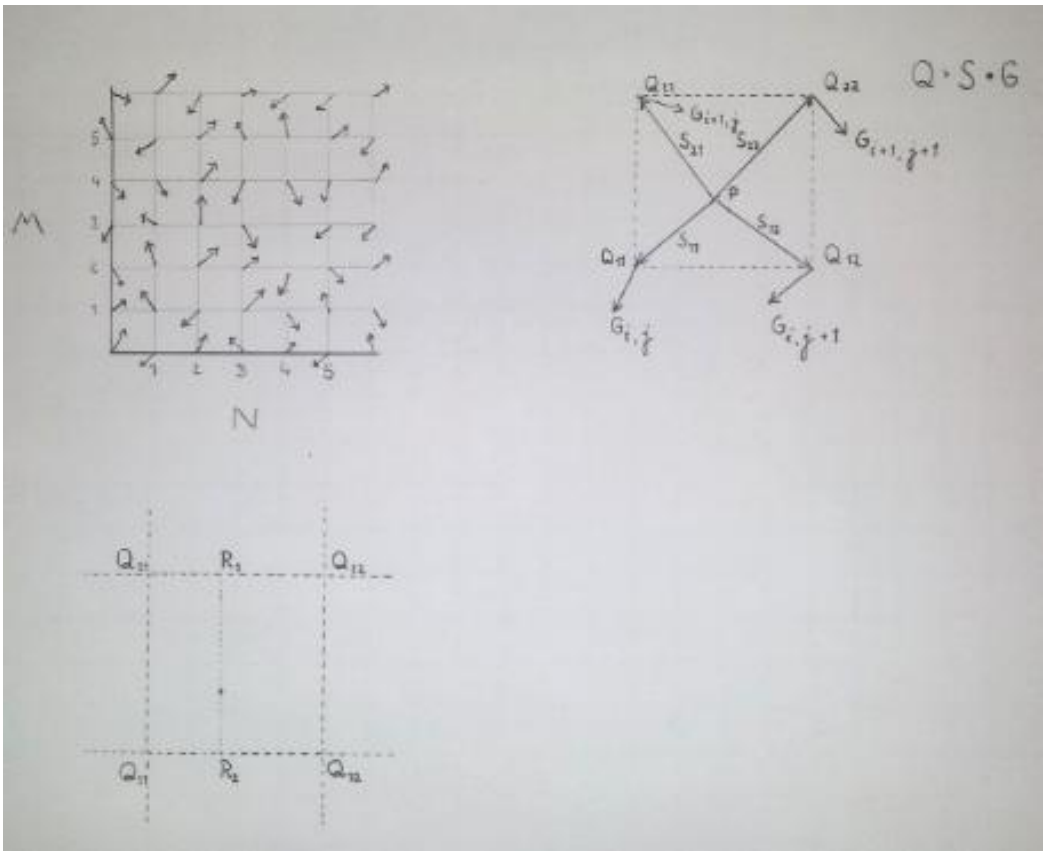


KUVA 18. Tekstuurit

7.7.2 Esimerkki

Seuraavaksi käsitellään yhtä Perlinin kohinan tuottamisen menetelmää. Tarvittavien vektorimatematiikan operaatioiden tarkat yksityiskohdat ovat tämän kuvailun ulkopuolella, mutta algoritmi ei edellytä pistetulojen laskemista ja interpolointia monimutkaisempia operaatioita.

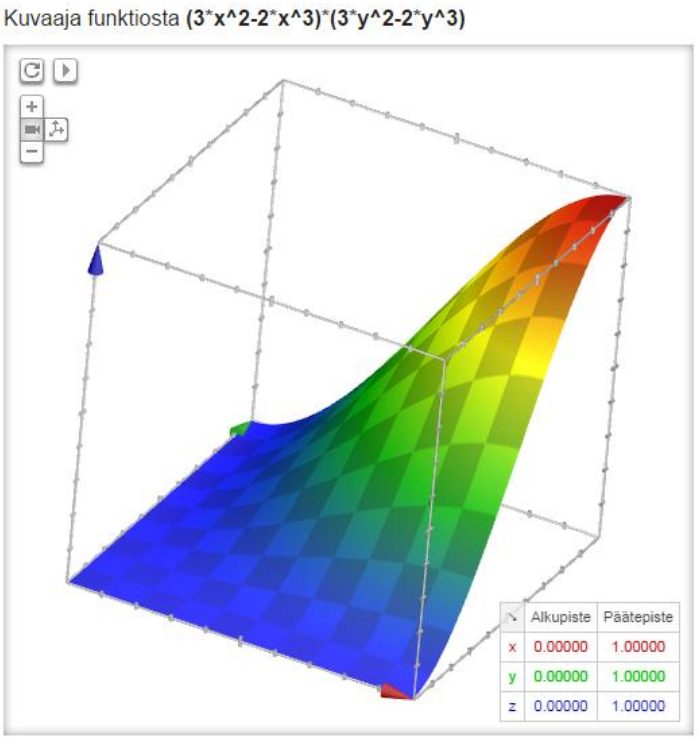
Aloitetaan alustamalla kaksiulotteinen taulukko satunnaisvektoreilla. Satunnaisvektorit on merkitty kuvaan 12 kirjaimella ”G”. Tämän jälkeen taulukosta otetaan näytteitä halutulla resoluutiolla. Esimerkin näytepiste on merkitty kuvaan kirjaimella ”P”. Näytepisteelle luodaan neljä vektoria (merkitty kuvaan kirjaimella ”S”), jotka vastaavat matkaa ja suuntaa näytepisteestä ”P” kuhunkin ympäröivään satunnaisvektoriin ”G”. Kustakin vektoriparista ”S” & ”G” otetaan pistetulo (merkitty kuvaan kirjaimella ”Q”). Näytepisteen lopullinen arvo määräytyy interpoloimalla nämä pistetulot. Interpoloinnin voi tehdä lineaarisesti ottamalla pistetuloista keskiarvot horisontaaliseesti ja vertikaalisesti, mutta paremman lopputuloksen ja pehmeämmän muodon lopulliseen kuvaan saa esimerkiksi kuvassa 13. esitetyllä s-curve funktiolla.



KUVA 12. Perlinin kohinan laskeminen

Interpoloinnilla tarkoitetaan, että kahden tunnetun tarkan lukuarvon välille määritetään arvioituja lukuarvoja jollakin tietyllä säännöllä. Jos esimerkiksi tiedetään jonkin kappaleen sijainti kahdella ajanhetkellä, voidaan interpoloimalla määrittää kappaleen arvioidut sijainnit ajanhetkien välillä. Mikäli kappaleen oletetaan liikkuneen tasaisella nopeudella, kutsuttaisiin interpolointia lineaariseksi. Koska edellä kuvatun näytepisteen arvo on näytepistettä ympäröivien satunnaisten vektorien ja näytepisteestä kulmapisteisiin piirrettyjen vektorien pistetulojen painotettu keskiarvo, ongelmaksi jää enää tämän keskiarvon laskeminen siten että se ottaa huomioon myös pisteen sijainnin.

Kuvassa 13 esitetty kolmiulotteinen funktio vastaa tähän ongelmaan. Jos kyseinen funktio rajataan alueelle 0 – 1 kaikilla akseleilla, se muistuttaa sulavasti kaartuvaa rinnettä. Sen keskeinen ominaisuus on siinä, että jos asetetaan päällekkäin neljä tällaista rinnettä, joita kutakin on käännetty siten, että niiden korkein kohta on eri kulmassa ja lasketaan ne yhteen, tuloksena on tasainen pinta, jossa Z saa arvon 1 kaikilla x:n ja y:n arvoilla. Tällaista funktiota voidaan hyvin käyttää Perlinin kohinan interpoloinnissa.



KUVA 13. Kohinakuvien interpoloinnissa käytettiin kuvan s-curve funktiota. Kuva on tuotettu Googlen graafisella laskimella.

8 KÄYTTÄJÄTESTAUS

8.1 Käyttäjätestauksen tausta

Pelille toteutettiin käyttäjätestaus 26.8.2014 osana laajempaa *Galaxin* palveluihin liittyvää käyttäjätestausta. Ensin testattavina oli kymmenen noin kymmenvuotiaan pojan, sitten kymmenen tytön joukko. Heille esitettiin ensin *Galaxin* ohjelmia ja heidän reaktionsa videoitiin. Testattavien reaktioita seurattiin tarkkaamosta käsin. Tämän jälkeen heitä kierrätettiin pienemmissä ryhmissä eri pisteillä, joilla heidän tuli käyttää tai testata aiheeseen liittyviä palveluita.

Testausasetelmaa ja jopa pelin toiminnallisuuksia varioitiin hieman testauksen aikana, jotta testauksesta saataisiin mahdollisimman suuri hyöty. Testaus aloitettiin esittelemällä peli lyhyesti, kertomatta kuitenkaan sen yksityiskohdista tai toiminnasta. Testaajien annettiin sitten kokeilla peliä ja pyrkiä läpäisemään kaksi pelin ensimmäistä kenttää. Testausta varten laadittiin myös kirjallinen ohje pelin kontrolleista. Vastaava ohje on liitetty peliin myöhemmin (Kuva 19), mutta sitä ei testauksen ajankohtana ollut vielä toteutettu.

Lapset pelasivat peliä innokkaasti ja antoivat siitä myönteistä palautetta. Palaute on testausolosuhteista johtuen epäluotettavaa, mutta signaalia voi pitää positiivisena. Pelin hyvinä puolina mainittiin “tutustuminen”, “kokeileminen”, “räjäyttäminen”, “strategian miettiminen”, “itse tekeminen” ja “grafiikat”. Peliin toivottiin lisää “hahmoja”, “varusteita”, “kaverin kanssa pelaamista” ja “salakäytäviä”.

8.2 Hahmon liikuttaminen

Lapset osasivat liikuttaa pelihahmoa pääsääntöisesti varsin hyvin. Kaksi seitsemästä osasi liikkua hahmolla heti erittäin taitavasti, yhdellä oli liikkumisessa vaikeuksia, mutta hänkin selvisi ensimmäisistä tasoista. Peliin on toteutettu kahdet osittain rinnakkaiset tavat liikkua (Kuva 19). Ratkaisu osoittautui järkeväksi. Pelejä vähemmän pelanneet löysivät helpoiten nuolinäppäimillä tapahtuvan liikkumisen, tottuneemmat pelaajat käyttivät W,A,S,D nappuloita, joiden viereen pelin esineiden pikanappulat on sijoitettu.

Pelin vaikeusaste näyttäisi olevan suuresti yhteydessä hahmon liikuttamiseen. Suurin osa testaajista piti pelin vaikeusastetta sopivana, mutta testauksen tuloksena sitä laskettiin hieman, jotta palveltaisiin laajempaa yleisöä. Eritasoisia pelaajia kannattanee muutoinkin pyrkiä palvelemaan luomalla kentistä sellaisia, että ne ovat suoritettavissa kohtalaisen helposti, mutta riskejä ottamalla saavutettavissa olisi jännittävämpi kokemus. Kenttäsuunnittelu voi ottaa tämän huomioon esimerkiksi sallimalla toisinaan vaikean kohdan kiertämisen esimerkiksi kaivamalla reitin sen ali.

8.3 Valikoissa liikkuminen

Peli aukeaa näkymään, josta pääsee aloittamaan pelin, luomaan oman kentän tai syöttämään kenttäkoodin. Tässä näkymässä pystyy kuitenkin jo liikuttamaan pelihahmoa. Tämä aiheutti sen, etteivät kaikki heti klikanneet “Aloita uusi peli” nappia. Joidenkin testattavien kohdalla tämä hidasti testauksen alkua, mutta aloitusruutu säilynee tällaisena. Se tarjoaa turvallisen harjoittelupaikan pelin kontrolleille ja esittelee pelin ominaisuuksia heti ensimmäisessä näkymässä.

Hieman yllättäen useat lapset eivät mieltäneet Esc-nappulaa luonnolliseksi tavaksi liikkua valikoissa taaksepäin. Taitaakin olla niin, että Esc:n merkitys on huomattavasti vähemmän tuttu nuoremmalle pelaajasukupolvelle. Se ei tule tutuksi pelaajille, jotka pelaavat pelejä pelkästään konsoleilla, puhelimella tai selaimessa. Esc-navigoinnin rinnalle toteutettiin tämän huomion vuoksi x-symboliin perustuva nappula.

8.4 Oman kentän luominen

Lapset osasivat käyttää kenttäeditoria varsin hyvin. He osasivat piirtää kenttään maata, asettaa siihen maalipaikan ja pelihahmon sekä testata tekemäänsä kenttää. Kentän jakaminen oli heille odotetusti hieman vaikeaa. Tämä on erityisen ymmärrettävää sillä testauksen alussa pelistä oli käytössä versio, joka edellytti kentän nimeämistä, kuvauksen kirjoittamista ja tekijän nimen kirjoittamista. Nämä ominaisuudet otettiin pois kolmelta viimeiseltä testaajalta ja suoritus helpottui selvästi. Mikäli näitä ominaisuuksia myö-

hemmin halutaan käyttää, tulee ne toteuttaa asetettaviksi ennen jakamis-napin painamista. Jakamisen tulee olla äärimmäisen helppoa.

Kenttäeditoriin on testauksen jälkeen tullut useita uusia ominaisuuksia, joiden käyttöä ei ole testattu kohderyhmällä. Palautetta niiden toiminnasta voidaan kuitenkin olettaa saatavan julkaisun jälkeen netin kautta.

9 POHDINTA

Kätkö on tällä hetkellä *Galaxin* netin selvästi suosituin sisältö. Näin ollen voidaan sanoa, että työn tavoite on saavutettu. Peli on saanut paljon positiivista palautetta sekä kehittämisehdotuksia, joista osa on jo toteutettu. Työ pelin parissa jatkuu ja sille on suunnitteilla lukuisia uusia ominaisuuksia. Myös monet nykyiset ominaisuudet pyritään tekemään modulaarisemmiksi ja monikäyttöisemmiksi.

Yksi tulevaisuuden tavoite on käyttäjien välisen interaktion mahdollistaminen. Tämä edellyttänee käyttäjätilien luomisen mahdollistavan järjestelmän toteuttamista ja itse tehtyjen kenttien jakamisen yhdistämistä sosiaalisen median toimintoihin. Jännittävä tulevaisuuden haaste on myös reaaliaikaisen moninpelin toteuttaminen. Tavoitetta kohti voidaan kulkea vaatimattomampien välitavoitteiden kautta. Tällaisia voisivat olla esimerkiksi pelin tilan tallentaminen, reaaliaikainen keskustelu ja vuoropohjainen kaksinpeli.

Uskon pelin tyyliin ja tavoitteeseen luoda mielenkiintoinen pelillinen kokonaisuus erilaisten pienten simulaatioiden kautta. Miellän tällaisiksi simulaatioiksi kaikki todellisen maailman ilmiöitä jäljittelevät toiminnallisuudet, kuten fysiikanmallinnuksen ja tekoälyn sovellukset, mutta myös niiden primitiiviset alkuasteet. Tarinallisuus ja ennalta määrätyt tapahtumat luovat peleille kehystä, mutta vasta pelin simulaatioiden vaikeasti ennakoitavat, mutta luonnollisesti toisiinsa linkittyvät seuraukset ovat niiden sisältöä.

LÄHTEET

Bibeault, B. & Katz, Y. 2010. jQuery in Action, Second Edition. Greenwich, Connecticut: Manning Publications.

Brampton, M. 2010. PHP 5 CMS Framework Development, Second Edition. Birmingham: Packt Publishing.

de Carpentier, G. 2014. Research Article: Analytical Ballistic Trajectories with Approximately Linear Drag. International Journal of Computer Games Technology. Volume 2014, Article ID 463489. Cairo: Hindawi Publishing Corporation

Lindley, C. 2012. JavaScript Enlightenment, Edition 3. California: O'Reilly Media.

Mozilla Developer Network. 2014. JavaScript Overview
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/JavaScript_Overview#JavaScript_and_the_ECMAScript_Specification

Osmani, A. 2014. Learning JavaScript Design Patterns, Volume 1.6.0. California: O'Reilly Media.

Shiffman, D. 2012. The Nature of Code. Version 0005. http://natureofcode.com/book/

Yap, P. 2002. Grid-Based Path-Finding. Advances in Artificial Intelligence.
Lecture Notes in Computer Science Volume 2338, 2002, pp 44-55.

World Wide Web Consortium. 2014. HTML5 A vocabulary and associated APIs for HTML and XHTML. W3C Recommendation 28 October 2014. http://www.w3.org/TR/html5/