Säikeistetty grafiikkamoottoriprojekti (ThreadedEngine)

Sakari Tanskanen
Miika Lehtimäki
Eero Heikkinen
Antti Hatanmaa

Kommunikaatio

IRC-kanava #ThreadedEngine @ IRCnet

Git-repositorio

https://github.com/EeroHeikkinen/ThreadedEngine

1 Johdanto

Tarkoituksena on luoda mahdollisimman johdonmukainen, säikeistetty OpenGL-grafiikkamoottori. Moottori on tekijöilleen ensisijaisesti oppimisprojekti, jota voi kurssin jälkeenkin käyttää mm. erilaisten peli- tai demoprojektien pohjana. Tämän vuoksi sen tulisi olla riittävässä määrin modulaarinen, jotta se mahdollistaa spesifisempien komponenttien erilaisten toteutustapojen testaamisen ilman suuria rakennemuutoksia. Liika granulaarisuus ei tästä syystä myöskään muodostune ongelmaksi, sillä moottorin ominaisuuksia voi myöhemmin käyttökohteesta riippuen myös rajoittaa.

2 Rakenne

Ohjelma tulee alustavasti koostumaan viidestä ylätason komponentista, jotka ideaalitilanteessa toimivat omissa säikeissään ja jotka on myös nimetty sen mukaan. Alemman tason komponentteja kutsutaan entiteeteiksi, ja sellaisia ovat esimerkiksi maasto, maastossa oleva kivi jne.
Kaikki komponentit sisältyvät kaikenkattavaan Device-olioon, joka luodaan piirtämisen alussa ja tuhotaan sen päätyttyä.

2.1 Ylätason komponentit

Ylätason komponentit ovat:

1. ResourceThread
    - ResourceThreadin pääasiallinen tehtävä on palauttaa resurssiviittauksia entiteettien käyttöön.

- Pyrkii luonnollisesti aina palauttamaan viittauksen jo ladattuun resurssiin, eli lataa resursseja vain tarvittaessa.
- Hallitsee resurssien tuhoamista, mikäli niitä ei enää tarvita. (tätä varten voinee kehittää kaikenlaisia hienoja resurssinkäyttöanalyysialgoritmeja... :D)

2. EventThread

   - EventThreadin tehtävänä on hallita entiteettien välistä kommunikaatiota.
   - Alustavasti jokaisen eventin yksilöi merkkijono, jota entiteetti voi käyttää rekisteröityäkseen kuuntelemaan kyseistä eventtiä.
   - Entiteetti voi rekisteröityä kuuntelemaan tiettyä eventtiä kertomalla EventThreadille kyseisen eventin merkkijonon sekä halutun callback-jäsenfunktionsa, jota EventThread kutsuu eventin tapahtuessa.
   - Entiteetti voi luonnollisesti myös laukaista jonkin eventin kertomalla EventThreadille kyseisen eventin merkkijonon sekä oheisdatan, jolla tämän eventin callback-funktioita kutsutaan. (Oheisdata voi olla vaikkapa hiiriklikkauksen koordinaatit tai up/down-tieto näppäinpainalluksesta.)
   - SFML-eventtejä voitaneen käyttää joidenkin eventtien pohjana, mutta toteutusta ei sidota SFML:ään.
   - std::map merkkijonoille ja callbackeille?

3. RenderThread

   - Vastaa ainoana säikeenä kommunikaatiosta grafiikkakortin kanssa. (GLcontext on rajattu yhteen säikeeseen kerrallaan.)
   - Tutkii Devicen sisältämää graafia piirrettävistä objekteista ja niiden suhteista. ja vastaa piirtämisestä. [1]
   - Sisältää mahdollisesti useampia renderöijiä, jotka voivat piirtää eri asioita, tai asioita eri tavalla. (Esimerkkinä vaikkapa 2-ulottein HUD tai selektiivinen raytracer, jota käytetään vain joihinkin osiin näkyvää aluetta.)
   - Entiteetit voivat vaikuttaa piirtämisen toteutukseen RenderThreadin tarjoaman rajapinnan avulla. (Yksinkertaisena esimerkkinä player-entiteetin on voitava määrätä käytetty kameramatriisi.)

4. PhysicsThread

   - Vastaa objektien sijainnin päivityksestä Devicen graafissa.
   - Toteuttaa rajapinnan, jonka kautta mm. logiikka voi saada tietoa fyysisestä ympäristöstä.
   - Toistaiseksi sijaintien päivitystä lukuunottamatta vähiten tärkeä komponentti. Myöhemmin pyritään todennäköisesti sisällyttämään tähän komponenttiin jokin valmis fysiikkamoottori.

[1] Tämä graafi tulee todennäköisesti olemaan vaikein yksittäinen osa moottorin toteutusta. Sen on kuvattava maailma tehokkaasti, ja lisäksi sen on toimittava oikein samaan aikaan kun useampi säie tutkii ja yksi muokkaa sitä.

5. LogicThread
    - Vastaa yleisesti ohjelman kulusta sekä sen sisäisestä logiikasta.
    - Käyttötarkoituksesta riippuen mm. tekoäly kuuluu tälle komponentille.

2.2 Entiteetit

Entiteetit tulevat olemaan moottorin kaiken toiminnallisuuden perusta. Entiteettejä voivat olla esimerkiksi maailma (esimerkiksi maasto ja skybox), pelaaja, vihollinen jne., ja ne sisältävät toiminnallisuuskomponentteja, joita voidaan listata tietorakenteisiin toisistaan riippumatta. Tällä pyritään siihen, että kaikkien säikeiden ei tarvitse käydä kaikkia entiteettejä läpi jatkuvasti, sillä on mahdollista olla mm. staattisia piirrettäviä entiteettejä, joita fysiikka- tai logiikkasäikeen ei tarvitse huomioida, tai jopa puhtaasti loogisia entiteettejä.

Entiteettien rakenne voidaan toteuttaa ainakin kahdella eri tavalla:

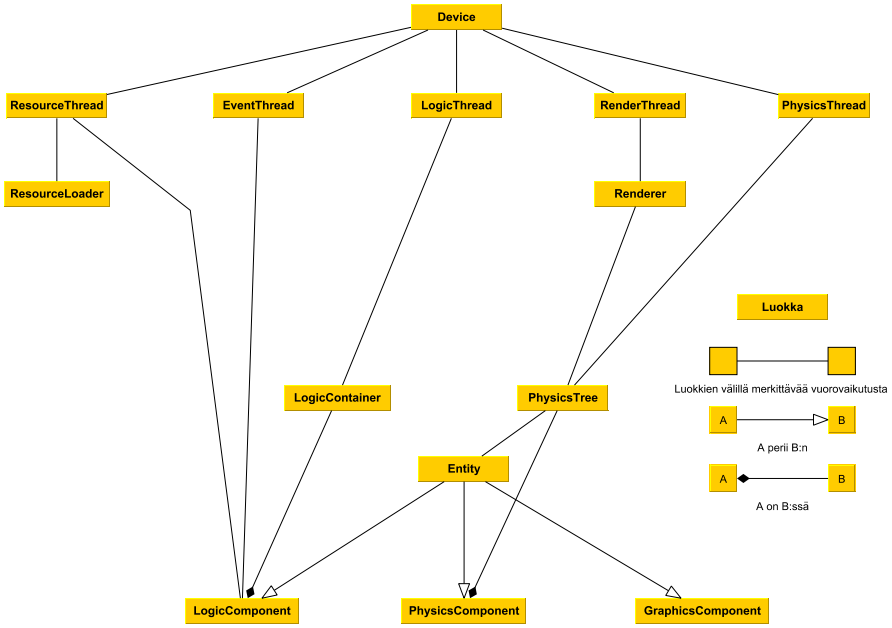
1. Entiteetit voivat periä ominaisuusluokkia (moniperintä)
2. Entiteetit sisältävät jäseninä ominaisuusolioita.

Periaatteessa molemmilla pitäisi päästä samaan lopputulokseen, mutta ensimmäinen vaikuttaa äkkiseltään johtavan yksinkertaisempaan koodiin. Timanttiongelmaa ei pitäisi ilmetä, sillä entiteetit eivät itse harrasta moniperintää eivätkä ominaisuusluokat peri. Mikäli ongelmia kuitenkin ilmenee, voidaan toteutustapa vaihtaa.

Esimerkkejä ominaisuuksista:

- Perustason spatiaaliset ominaisuudet kuten paikka ja orientaatio.
- Graafiset ominaisuudet kuten mesh/sprite/partikkelit.
- Fyysiset ominaisuudet kuten nopeus ja massa.
- Loogiset ominaisuudet kuten tekoäly.

Alustava luokkakaavio:



3 Työnjako

Antti ja Miika:
Device sekä RenderThread melko tiiviissä yhteistyössä. Nämä ovat myös sen verran vaativia komponentteja, että myöhemmin muutkin siirtynevät auttamaan näiden toteuttamisessa.

Sakari:
LogicThread sekä toissijaisesti PhysicsThread. Tähän kuuluu varovaista tutustumista mahdollisiin fysiikkamoottorikandidaatteihin, jotta fysiikkasäieobjektin implementaatio sekä rajapinta vastaavat fysiikkamoottorin tarjoamia mahdollisuuksia.

Eero:
EventThread.

4 Aikataulutus

Projektia tullaan pitämään mahdollisimman paljon toimivassa tilassa ja ominaisuuksia lisätään yksitellen niin paljon kuin aikaa(ja mielenkiintoa) riittää. Prioriteettilistalla ovat ensimmäisenä renderöinnin ja logiikan toimimaan saattaminen, jonka jälkeen käyttäjän syöte ja fysiikkamoottorin liittäminen. Resurssinlataus voidaan tarvittaessa jättää kovakoodatuksi, joten se implementoidaan perusominaisuuksista viimeisenä. Lopussa olisi tarkoitus panostaa erilaisten renderöintitekniikoiden implementointiin.

5 Muita huomioita

Käytetyt kirjastot:

- Käyttöjärjestelmärajapinta: SFML
- Grafiikka: OpenGL
- Fysiikka: Bullet?