Narvik University College

# Game Design
# STE6274

Pål Solberg Trefall

[a] *Narvik University College, Lodve Langesgt. 2, P.O. Box 385, N-8505 Narvik, Norway.*

11. October 2011

## Abstract

This document is a report of the task, the work, insight and the results from the STE6274 project in Game Design at the University College of Narvik, Norway, 2011. Through the project a client-server based game was made that would rely on client-server negotations handling and proper definition of responsibility in the client. While the students coded one client implementation each, the server was provided by the teachers. The client presented in this report focused heavily on the architectural aspects of the assignment. A component-based entity system lay the basis for the game-layer design, while the network-layer was heavily dictated by Qt types and methodology. The result is a top-down space-shooter that sets each client in control over a spaceship. Static scenery objects defined by the server is represented by space-stations.

## 1. Introduction

First half-semester of fall 2011, the project in Game Design was handed out. I saw a great opportunity through the assignment to test and improve on an entity system library I had been writing over the summer, called Totem EDK, and this lay the basis for the game-layer of the client. Unfortunately, a client is supposed to be as dumb as possible, so which parts of the architecture that could be divided into components of logic was quite sparse. It lay the basis for a highly maintainable code-base however, with clearly divided modules of logic working together in harmony. There was also the requirement that we use Qt, and the GraphicsView API. This put some hard restrictions on the implementation that, for me, really hurt progress and what I could do with the limited time. Especially the GraphicsView API, which forces the OpenGL code into an integrated OpenGL2 paint engine. Qt also has some custom compilation steps that's required for a lot of it's more advanced features, like signal/slot, to be used. I decided to try and integrate the project definition in Cmake, and let Cmake handle all the Qt custom compilation for me, instead of writing custom compile-steps into the Visual Studio project files. As I came to realize, this lay the basis for really good architecture, seperating project-file definition from project code.

## 2. Material & methods

### *2.1 Networking*

The assignment involves client-server network communication via a defined game protocol. As the server was being developed in parallel with the assignment, we had to rely entirely on the protocol at times, without the ability to ping-pong features to server.

#### *2.1.1 Client*

The client was implemented on top of QtcpSocket and the game protocol defined for the assignment. First, I attempted several multi-threaded approaches to using QtcpSocket, as it's a great candidate for running in parallel with the rest of the application as long as the bridge between the socket handling and the application-layer is secured via a queue with locks. I tried both with QthreadPool and Qthreads, but to no avail. There was always some weird, mystical Qt warning or error that stopped me from doing it the way I wanted to.

Having implemented multi-threaded servers and clients before, I felt that Qt forced my hand a bit too hard with how heavily it was checking up on me. In the end, I had to fall back to running the socket handling in the main thread, along with everything else. I kept a lot of the design however, holding one task object for parsing and one for packing network packets.

### *2.1.2 Parsing packets*

When parsing incoming server packets, the parser always makes sure that there's enough bytes available on the socket before attempting to read a packet. It first reads the header, then based on the info in that header, it reads the body of the packet. The parsed server packet is then queued on a list. Every frame/tick, the client iterates over all newly parsed packets and handle them.

### *2.1.3 Packing packets*

When packing packets, the packer relies on the template implementation to write data on the byte array stream. When a packet is packed, it's queued on a list. Every frame/tick, the client iterates over all newly packed packets and sends them on the socket to the server.

## *2.2 Qt*

Qt was originally created by Trolltech, but is now owned by Nokia under the name Qt Development Frameworks. It's a cross-platform platform and user-interface framework. It holds a signal/slot api for event handling which makes it easy to link functionality with user interface interaction, a complete network api, threading api, and much more.

In this project, Qt was used for windowing, OpenGL paint engine of the game scene, user interface, input handling, networking, multithreading, some serialization (Qstring's .arg functionality) and the signal/slot api to communicate mainly between network layer and the graphical user interface.

Qt is a very heavy library to work with. It's not just a maze of different kinds of functionality, but it's also coded in such a way that it doesn't feel familiar, it doesn't follow the std standard way of design that I'm so used to, and Qt goes a long way to make sure you use it the way it wants you to (little room for approaching a solution in your own way, at least in my experience). Thankfully the documentation is very good, so once you know about a problem, there's no problem to look up the documentation that explain why it's so. Then again, I've found that I usually only find out Qt didn't accept my approach after several hours of coding...

## *2.3 OpenGL*

I decided to use OpenGL and render the game scene on my own, or as much of it as Qt's QgraphicsScene object would allow me to. Unfortunately, the QgraphicsScene force the use of a paint engine backend, and Qt only bundles an early OpenGL2 painter. I tried to create an OpenGL3 context and just maybe Qt would allow me to write OpenGL3.3 code in my game-layer. Unfortunately this was not the case, and I was stuck with immediate-mode rendering for the duration of this project.

I used the component-based approach to add rendering functionality on game entities. The Renderable component would take a RenderSystem reference by it's constructor and use this relationship to render the entity that owned the component. In the case of skybox and particle rendering, I implemented those in separate components, and registered a delegate, which the renderable would call if available on the entity, which would handle custom render functionality.

## *2.4 Totem Entity Development Kit*

The Totem EDK was used quite extensively in the game-layer of the client, and is a library I've developed over quite some time. Not until this summer did I start to code on it with the intention of making it a robust, fully featured and user-friendly entity development kit. It's still under development, and this project helped me improve it a lot. Internally it showcases advanced C++ topics, such as template programming, preprocessor macros (used very carefully), signal/slot events, delegate functions, function pointers, *any* variable usage, polymorphism, string hashing for optimizing string key lookup speed in maps and lists, doxygen documentation, factory pattern, component-oriented programming, template-based property system that use the pimpl-pattern together with a shared-pointer on the data, property serialization and deserialization.

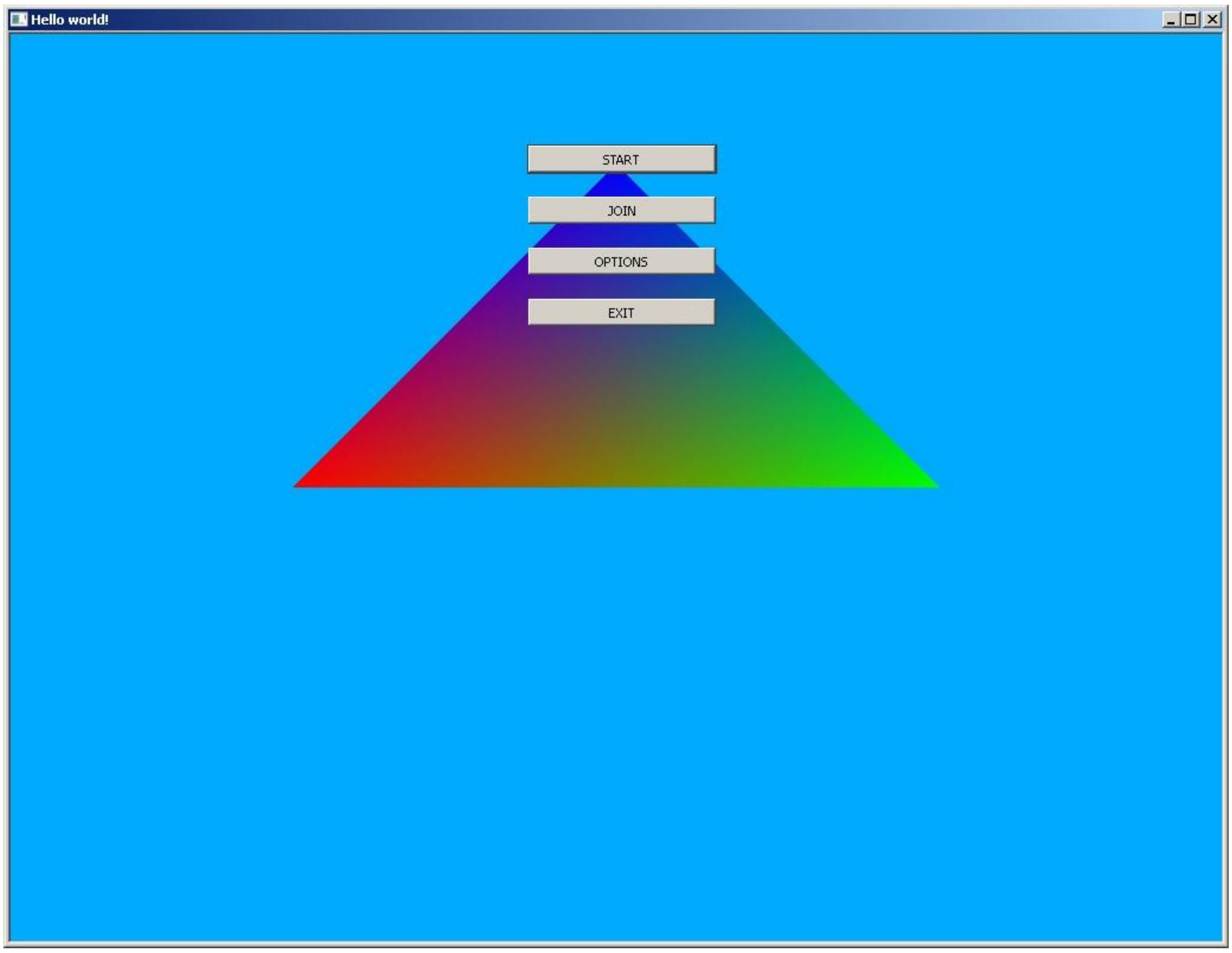
The Totem EDK makes definition of entiy logic and data simple, flexible and maintainable. Logic is stored in modular components of logic, providing the designer with plug 'n play logic modules when defining entities. The goal of the approach is to flatten the hierarchy and make logic and data manipulation modular instead of hierarchial.

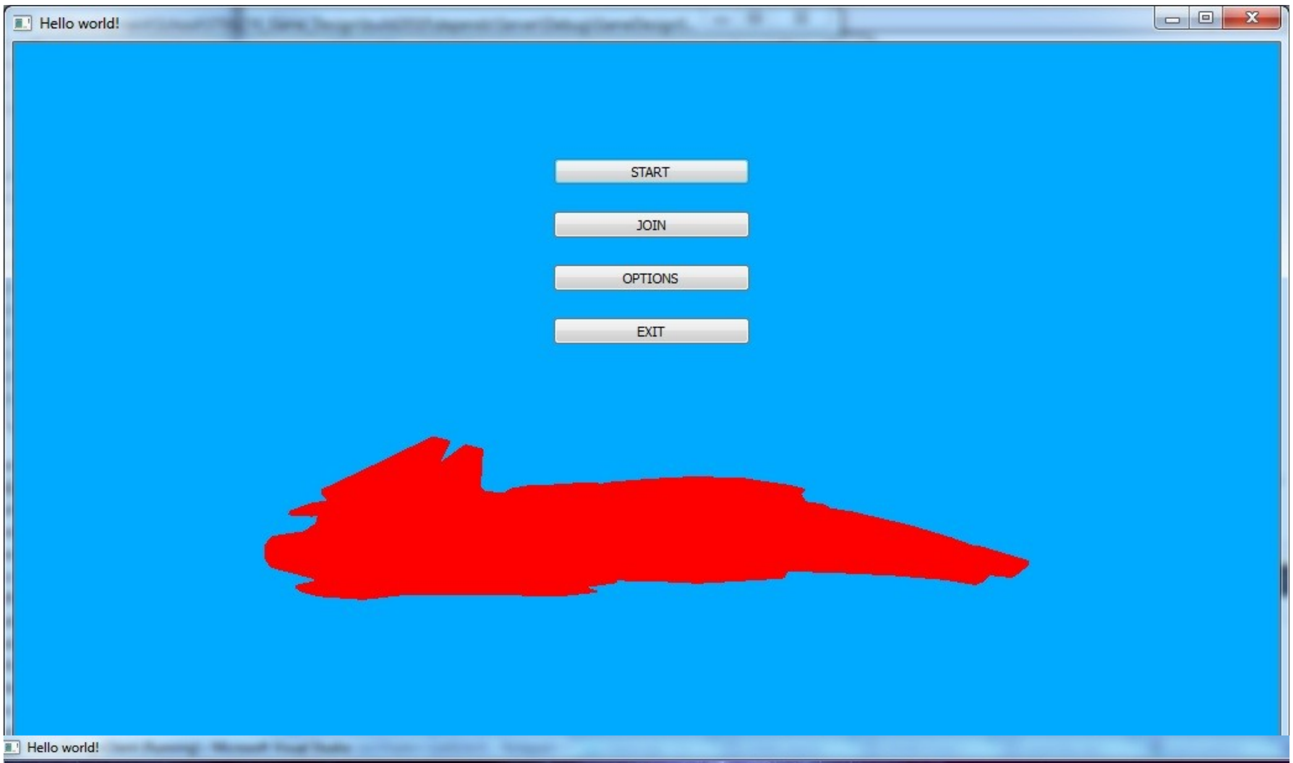
### *2.5 Application-loop*

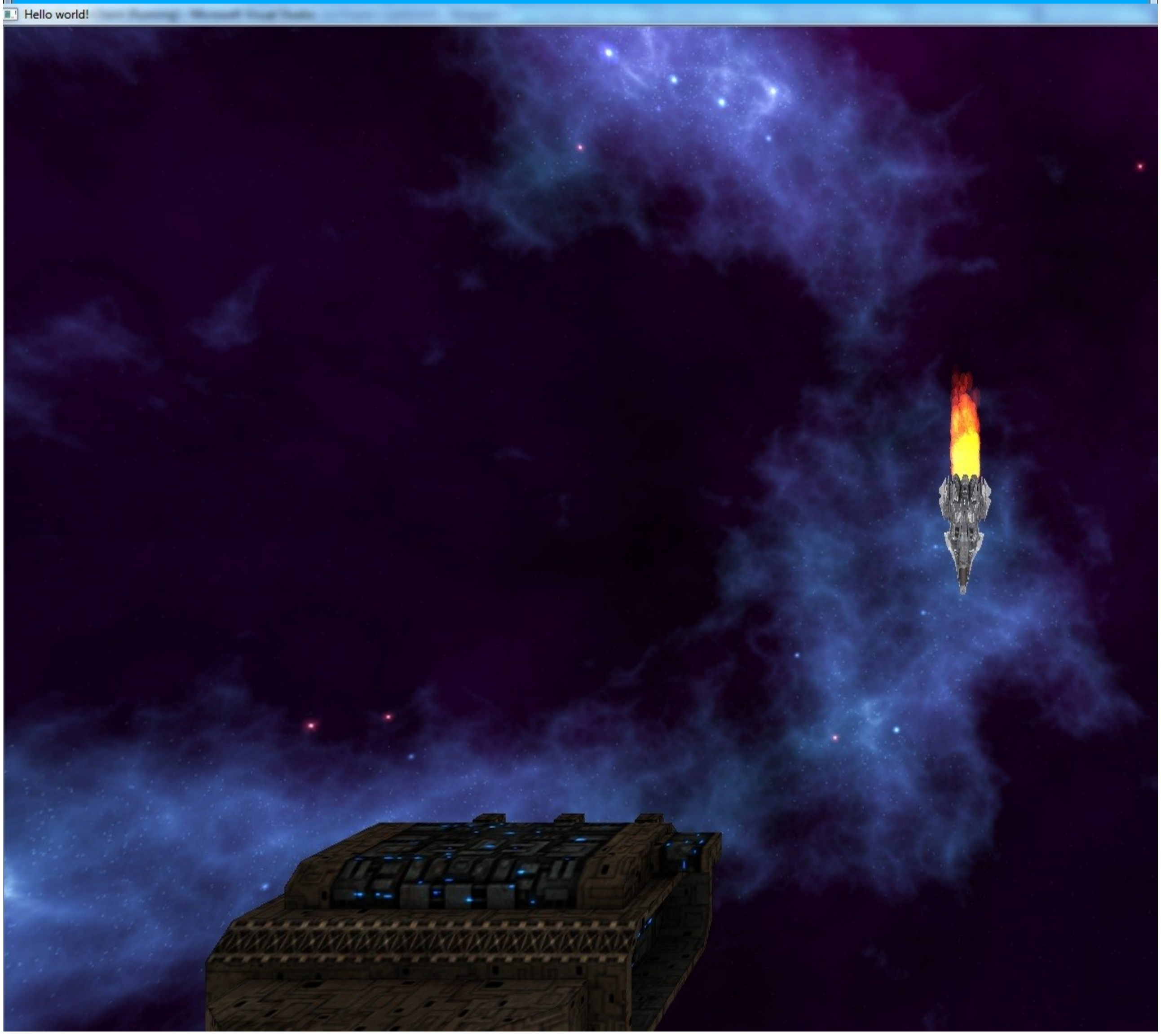
The application loop start's in QgraphicsScene's drawBackground. It calls the Game class' advanceFrame, which in turn extracts parsed game packets, updates the client (network-layer) and entity manager, compiles the render system and finally renders.

## 3. Results

Following is presented some screenshots from the game.




Hello world!
START
JOIN
OPTIONS
EXIT


Hello world!

## 4. Conclusions

Though I never quite reached all the goals that I set out with, I felt that the most important parts I wanted to focus on with the project was completed quite well.

I got to challenge my Totem EDK and write an entire Game-layer (although a dumb game layer due to client restrictions) based on the component oriented paradigm, and I got to play around quite a lot with sockets and thread-pools, even though the last one didn't end up in the final assignment project files, I still learned a lot from it.

I thought it was a great challenge that the server was programmed at the same time as the client. It created a more realistic working environment, where you can't always expect to only work against robust and tested code. It forced me through some debug grinds, which I learned from.

Qt didn't really work well for me though. I know that had I just been allowed to use more familiar APIs, I would have been able to do so much more. Qt lay it's iron-regime on socket handling, threads and rendering, which pretty much sums up this assignment's requirements.

In the end, it's been a great experience to work on this project, and it was great to see networking and threading being a requirement of the assignment.

## Acknowledgements

The author would like to thank his fellow students for sharing know-how and ideas on this complex project. Also a big thank you to my wife, who took care of our baby while I sat long hours on this assignment.

## References

Totem EDK, 2011, Entity-Development-Kit, http://code.google.com/p/propcomp/ (12.10.2011).

Qt, Middleware, http://qt.nokia.com/ , (12.10.2011).

AssImp, 2010, Asset Import Library Middleware, http://assimp.sourceforge.net/ , (12.10.2011).

Quagmire Particle System, 2011, Particles, http://home.comcast.net/~kjmaz/ , (12.10.2011).

DevIL, 2011, Image Library, http://openil.sourceforge.net/ , (12.10.2011)

GLM, 2011, Mathematics, http://glm.g-truc.net/ , (12.10.2011)