OakVR Game Engine Concept

# Features

* Possibility to edit the game world from inside it
* Rapid iterative development process
* Scalability of the engine (both data and features)
* Portability?

# VR interface used in the game development process

Use Oculus Rift (or other VR devices) in order to have the developers inside the game as it is developed (You can think of constructing in Minecraft as an example). Add objects, edit their behavior, create events, place sounds, sculpt shapes and create experiences in a much more involved fashion than it is currently done.

# Rapid iterative development process

~~Client server structure of the engine during development would permit the compilation of small modules that would then be loaded by a program that disconnects from the main engine and connect after linking with the new functionality. This way the engine would continue running while new functionality is being added to the game. Real time editing of game objects would also be possible.~~

Functional and data modules that can be unloaded, recompiled, reloaded with state saving. (This will only work on platforms that support dynamic linking)

# Scalability of the engine

The engine should be able to support huge game worlds, any number of game assets and it should support adding new functionality easily to it. This would mean taking a modular approach to game engine features as opposed to a monolithic one.

# Portability?

For now the engine will only be developed with portability in mind but no effort will be taken to make it work on other platforms than Windows desktops for now.

# Engine components

## Core

This is the main component of the engine. It contains a resource manager, io wrappers and a server to which the editor can connect.

## Feature Modules

Modules encapsulate functionality like rendering, usable components, game components. Every module can be compiled and linked at runtime to the Connector module of the engine (described below). Through the client module, new functionality will be available to the running game at runtime. These modules can also link to the game directly, but they cannot be modified at runtime (to be used for the end of the development process).

## Connector Module

When the editor starts up a game (or game in development), both the game executable and the Connector module are started up, with the Connector executable connecting to the game executable. The Connector can be disconnected~~, linked with new or updated functionality~~ and be reconnected to the game in order to observe the changes.

## Editor

Manager for development resources, starts up the game and the connector module.

Its components are also used inside the game when it is running…. Editor module? For example the C++ editor should be available in the VR editor for changing the behavior of game elements.

# Modules breakdown

Below is a list of all the modules that need to be worked on at the beginning (what is foreseeable now). CP stand for cross-platform and PA stands for platform agnostic.

**Core**

* + StartUp (CP)
  + ResourceManager (PA)
  + FileIOWrapper (CP) (until File IO is introduced into the C++ standard language)
  + MathLib (CP)
  + NetworkWrapper (CP) (until Network functionality is introduced into the C++ standard language)
  + ModuleLoader
  + EngineBackbone(PA)
  + EditorServer (PA)

**Editor**

* + EditorGUI
  + EditorC++Parser (VS integration? Scintilla?)

**Connector** **Module**

* + EditorClient

**Feature** **Modules**

* + Renderer
  + Font
  + FormatLoaders(jpg, png, bmp, etc.)
  + KeyboardInput
  + MouseInput
  + GamepadInput
  + VRInput
  + ZipExplorer

# Module structure

A module is a logical representation of similar content, containing both classes and objects, which helps to structure a game and helps with the iterative development process.

A module has a name, a collection of registered entity classes and a collection of objects.

When a module is loaded (Dll/so/dynlib) it registers with the main application and loads the state of all its registered objects.

When a module is unloaded, the state of its objects is saved and it is deregistered from the main application.

Communication with the modules

OakVR

* Module2
  + Entity1
  + Entity2
  + Entity3
* Module1
  + Entity1
  + Entity2
  + Entity3

Each module entity is derived from Entity. At definition each Entity registers itself with the module it is defined in. Thus each module is capable of offering information about and of instantiating every instance from within. Each module that is linked to OakVR registers itself and gives access to all the entities in the module.

Entity registration

For each entity, a ReflectionInfo object will be created which has the following structure:

struct ClassInfo

**{**

std**::**string name**;**

std**::**vector**<**std**::**string**>** bases**;**

std**::**vector**<**MemberInfo**>** member;

**};**

struct MemberInfo

**{**

std**::**string name**;**

size\_t size**;**

size\_t offset**;**

**};**