OnLooker Game Framework

Version 3.0

Revision List

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Comments |
| 1 | Nathan Hanlan | 25/2/2014 | Initial Version |
|  |  |  |  |

**Table Of Contents**

1. Overview

a. Language

b. Hardware

c. Drivers

d. OS

e. Third Party

f. Core Systems

2. File Naming

3. Core Systems Designs

a. Math Library

b. Input System

b. Physics System

c. Rendering System

e. Time System

f. File Manager

g. Random Number Generator

h. Memory Management System

i. Resource Manager

j. World Manager

k. Networking

Link to Google Docs

<https://docs.google.com/document/d/182hktcm1VlHJp1Rofd51O3M3qymGH88lOSvZP3wucug/edit>

**[Page Left Blank]**

**1. Overview**

**a. Language**

The bulk of the code will be written in c/c++.

There will be language bindings for java for Android development.

There will be language bindings for Objective-C for IOS Development.

There will be language bindings for C# for Android development and IOS

development.

**b. Hardware**

The primary target hardware for development is primarily PC

The secondary target is mobile.

**c. Drivers**

The graphics drivers this engine targets is for opengl.

**d. OS**

The primary target OS is Windows.

The secondary target OS is Android / IOS.

**e. Third Party**  
 Third party librarys are subject to change.

•Glew

•GLFW

•OpenAL

•Boost::Thread

Glew for opengl extensions.

GLFW for window creation.

OpenAL for audio.

Boost::Thread for threading.

**f. Core Systems**

These are the following core systems planned out for the Game Framework thus

far.

a. Math Library

b. Input System

c. Physics System

d. Rendering System

e. Time System

f. File Manager

g. Random Number Generator

h. Memory Management System

i. Resource Manager

j. World Manager

k. Networking

**a. Math Library**

The math library is to be written to be fast, and efficient and will depend on the memory management system. Math will be written for both 2D and 3D. Data types used in math will be int, float, and double.

**b. Input System**

The input system is written to be operating / hardware specific. The front end code will be designed to be cross platform with the primary and secondary platforms.

**c. Physics System**

Physics System...Considering box2D

**d. Rendering System**

Rendering System since were developing with OpenGL API. Checks will need to be put in place to determine which opengl version. The front end code will be designed to be cross platform with the primary and secondary platforms.

**e. Time System**

The time is a system to keep track and handle game delta’s.

**f. File Manager**

The file manager is a system that helps read/write files to the appropriate places.

**g. Random Number Generator**

The Random Number Generator that’s probably going to be used is the mersenne twister.

**h. Memory Managerment System**

The memory management system is responsible for allocating a giant block of memory and then handling the memory itself. \*\*\*Research\*\*\*

**i. Resource Manager**

The resource manager is a system responsible for loading in resources at appropriate times and providing a global access point to them throughout the program.

**j. World Manager**

The world manager is responsible for handling general data flow throughout the game loop.

**k. Networking**

The networking part is responsible for handling networking/multiplayer code. Data transfer between server and client.

**[Page Left Blank]**

**2. Naming**

**Commenting and Code Style**

**Commenting & Code style**

Functions Comment template:

/\*

\* Function: <Function Name>

\* Return Type: <Return Type>

\* Description: <Description of the functionality of this function>

\* Parameters: @parameter

\* Date Modified: <Date Modified (dd/mm/yyyy)> by <Author>

\*/

example:

/\*

\* Function: setX

\* Return Type: float

\* Description: Returns the x position of the rectangle

\* Parameters: @x - the x value you wish to set

\* Date Modified: 1/1/1988 by Nathan Hanlan

\*/

Class Comments template

/\*

\* Class: <Class Name>

\* Base Class: <Base Class Name>

\* Description: <Description of the class>

\* Date Modified: <Date Modified (dd/mm/yyyy)> by <Author>

\*/

example:

/\*

\* Class: Rectangle

\* Base Class: Object

\* Description: Holds vector and scale variables to store position and size.

\* This class also has various methods to access these variables and overloaded

\* overloaded operators.

\* Date Modified: 1/1/1988 by John Kloe

\*/

Header Comments template

/\*

\* Filename: <filename>

\* Date Modified: <Date Modified (dd/mm/yyyy)> by <Author>

\* Comments: <Version Number> <Comment>

\*/

example:

/\*

\* Filename: Rectangle.h

\* Date Modified: 1/1/1988 by John Kloe

\* Comments: v1.1.12 Added calcArea method and implemented.

\*/

**Variable Naming Guide**

Variables that are a parameter or argument are prefixed with the letter ‘a’ or ‘p’ to distinguish between the local variable and parameter or argument.

Example: int aMyParameterInt = 0; int pMyParamterInt2 = 0;

Global variables are prefixed with ‘g\_’ to distinguish global variables from the other variable types.

Example: int g\_MyGlobalInt = 0;

Static variables are prefixed with ‘s\_’ to distinguish them from the other variables.

Example: int s\_MyStaticInt = 0;

Constant variables are written in full capital letters and are spaced with underscores.

Example: int MY\_CONSTANT\_INT = 0;

Member variables are prefixed with ‘m\_’ to distinguish them from the other variables. Note static member variables are still prefixed with ‘s\_’ and not ‘m\_’.

Example: int m\_MyMemberInt = 0;

Function/method names start with lower case letters and then are camel cased afterwards.

Example: int myIntFunction = 0;

Class names start with Upper case letters and then are camel cased afterwards.

Example: class MyClass

Function Pointers are suffixed with the word ‘Callback’. Function pointers as a parameter or argument are prefixed with ‘cb’ as opposed to ‘a’ or ‘p’.

Example: void \* myFunctionCallback();

Example: void \* cbFunction();

**File Naming Guide**

Files are to be prefixed with the name OL\_. OL is for OnLooker.

Example: OL\_Texture.h / .cpp.

Files that are created to define enum’s are prefixed with the letter E after OL\_.

Example: OL\_EDirection.h.

Files that are created to define interface type classes (Abstract) are prefixed with the letter I after OL\_.

Example: OL\_IGameObject.h / .cpp.

**File System**

Project organization is key.

Project Directory -> Main.cpp. (The programs entry point).

Common Directory (Located outside of the project directory).

• Common -> Core (Contains the files core to the game engine and lower level programming. OS specific code).

• Core->OpenGL (Contains classes for rendering/ texture loading).

• Core->Physics(Contains classes for physics)

• Core->Window(Contains classes for window creation)

• Core->InputSystem(Contains classes for input systems)

• Core->ApplicationManager(Manages all of the data running during the

length of the application.)

• Common -> Math (Contains Math classes / structures and function for math

calculations)

• Math -> Vector2D

• Math -> Vector3D

• Math -> Matrix2x2

• Math -> Matrix3x3

• Math -> Matrix4x4

• Math -> Euler

• Math -> Quaterinion

• Math -> Functions

• Common -> Utilities (Contains utility classes / structures, definitions and

functions to help program).

•Utilities -> Conversion

•Utilities -> Debug - Console

•Utilities -> Debug - Log

•Utilities -> Random Number

• Common -> Game (Contains code for where all the game code is written)

•Game -> Game Class

•Game -> Additional Game Specific Files

• Common -> Libraries (Contains the location for all Libraries).

Resource Directory Located outside of the project directory).

• Textures

• Sounds

• Level Data

• Shaders

• Ini Files ( Engine Specific)

• Materials

• Models

• Scripts

• Mods

Commit Messages

List Changes:

Keywords: [Added / Implemented / Changed / Removed / WIP(Work In Progress)]

List Fixes:

List Known Bugs that require attention:

Example

Changes:

•Added & Implemented Entity System

Fixes:

•Fixed File Manager

Known Bugs:

•Enttiy System

**3. Core Systems Design**

**a. Math Library**

Define PI - 3.1459265358979323846;

Class Vec2(float, int, double) - 2 Dimensional Vector

Class Vec3(float, int, double) - 3 Dimensional Vector

Class Mat2x2(float,double) - 2 by 2 Matrix

Class Mat3x3(float,double) - 3 by 3 Matrix

Class Mat4x4(float,double) - 4 by 4 Matrix

Class Quaternion(float, double) - 4 Components for angle’s and rotation. Radians

Class Euler(float, double) - 3 Components for angle’s and rotation. Degrees

Function degToRad - Converts degrees to radians

Function radToDeg - converts radians to degrees

Function swap - Takes two variables and swaps their position;

**b. Input System**

Class Input - (Singleton Design) The front end programming where input can be checked

up on.

Class InputHandler - (Singleton Design) Registers Input from OS and sends it’s data to the Input Class

**c. Physics System**

Design, Rigidbooty

**d. Rendering System**

Design - System which renders with supplied vertices. (Immediate Mode, Vertex Array,

VBO/IBO).

Design - System which reads in texture data from disc.

Design - System which reads in vertex data from disc.

Design - System which reads in model data from disc.

**e. Time System**

Class Time - This class will handle the getting time from the OS and then send it to the

front end of the API so the user doesnt have to worry about it.

**f. File Manager**

Enum FileLocation - Defines File locations eg Images/Sounds

Function getPathFor(FileLocation) - returns the path in a string form for the file location.

**g. Random Number Generator**

Design a Mersenne Twister random number generator.

**h. Memory Management System**

Design - Block Allocators.

**i. Resource Manager**

Design - A Manager which loads resources in and handles the data appropriately.

**j. World Manager**

Design - A game class that implements functions of the game loop and handles core

engine data as well as game object data.

**k. Networking**

Research - Networking in C++.