Easy Game Engine

**TECHNICAL DESIGN DOCUMENT**

Version 1.0

Copyright © 2010~2012 LiCode Union.

目录

[Easy Game Engine 1](#_Toc343370139)

[1 引擎描述和功能介绍 3](#_Toc343370140)

[2 游戏功能模块简述 4](#_Toc343370141)

[3 通用游戏框架Framework 4](#_Toc343370142)

[4 引擎工作目录结构 4](#_Toc343370143)

[5 如何使用引擎 5](#_Toc343370144)

[6 FAQ 5](#_Toc343370145)

## 引擎描述和功能介绍

* 1. **开发人员**
  2. **引擎描述**

EasyGameEngine 针对于此，专门制定了自己的一套开发方针。

1. **跨平台编写方案**

使用native C/C++代码100%平滑过渡到android/ios/windows操作系统，未来将会支持更多的主流操作系统。

1. **涵盖各种功能模块**

游戏制作者不再需要大量的第三方库的支持，引擎包含各种常用功能模块，各种需求尽纳于此。

1. **游戏开发工具**

游戏制作者不再需要编写大量的代码，我们会制作各种各样的工具来简化游戏开发工作。

1. **通用游戏框架 Framework**

游戏项目间充斥着各种相似甚至相同的功能模块，此框架能解决各项目之间代码和资源重复率较高的问题。

1. **脚本方式开发游戏**

使用python脚本便能实现游戏客户端编写，非常方便，并且平台无关。

* 1. **功能介绍**

EasyGameEngine 是一款专门针对 2D/3D 游戏的跨平台游戏引擎，以易用性和跨平台特性作为重点，包括以下的主要功能模块。

* 1. **跨平台底层函数库**

支持各种平台的底层 API 和接口操作。

* 1. **数据库系统**

支持 SQL 数据基本操作。

* 1. **网络传输系统**

终端和服务器间的通信管理。

* 1. **2D/3D 图形处理系统**

2D 图形处理系统。

3D 图形处理系统。

* 1. **GUI 框架系统**

用户界面系统。

* 1. **脚本系统**

Python 脚本系统

* 1. **音频系统**

音频播放控制系统 。

* 1. **资源管理系统**

游戏资源管理。

## 游戏功能模块设计理念

* 1. **跨平台底层函数库**

此模块由两部分组成，一个是基础模块(Platform)，一个是接口功能模块(Engine)。

Platform主要以高内聚的实体类构成，目的是方便上层模块使用，所以绝大部分代码没有使用接口进行封装。这样子上层使用起来虽然和类型相关联，是紧耦合，但是此层作为引擎的最基础模块是利大于弊的。

Engine 则全部以接口的方式进行工作，并且提供各种复用性高的功能类（接口）。

* 1. **数据库系统**
  2. **网络传输系统**

此模块主要是以易用性作为出发点来设计，接受和发送数据处理都在一个线程里。

* 1. **2D/3D 图形处理系统**

此模块封装了各种底层渲染API(D3D,OpenGL,D2D,GDI)，并且使用桥接模式进行了隔离。

* 1. **GUI 框架系统**

此模块使用了组合模式进行构建，并且内部使用了策略模式进行GUI Components的管理。

* 1. **脚本系统**
  2. **音频系统**

此模块封装了各种音频库底层API(OpenAL,OpenSL)，并且使用桥接模式进行了隔离。

* 1. **资源管理系统**

## 通用游戏框架Framework

此模块主要用来简化各种游戏之间重复模块的开发工作，减少编码过程中的重复工作。

## 引擎工作目录结构

* 1. Build

批处理编译功能目录，用于制作版本的流程控制。

* 1. Document

引擎文档帮助目录。

* 1. EasyGameEngine

引擎Core 核心，底层模块功能目录

* 1. Framework

高层封装，依赖于底层功能模块的逻辑功能接口。

* 1. Tools

引擎工具目录

* 1. Tutorials

引擎功能示例目录

* 1. UnitTest

引擎功能单元测试目录

## 如何使用引擎

* 1. 如何初始化引擎
  2. 如何获得和使用基础功能接口
  3. 如何使用图形渲染
  4. 如何使用GUI渲染
  5. 如何使用网络功能
  6. 如何使用Framework高层模块功能
  7. 如何创建程序APP

## FAQ

* 1. 为什么使用RefPtr<>, 和 PassRefPtr<> ?

方便维护对象的生命周期，使用者不再用担心忘记调用Release() 等析构方法接口而导致内存泄露，并且

* 1. 为什么禁止使用多继承 ?
  2. 为什么不使用STL模板 ?
  3. 为什么内存泄露一定及时解决 ?
  4. 为什么在结束app(引擎析构前)要保证客户端的资源都被释放掉 ?
  5. 为什么禁止在客户端使用标准C 的功能函数 ?