Rayman

1. 程序设计

Rayman这款游戏是在JupiterEngine这套由我们编写的游戏API的基础上完成

的。JupiterEngine采用了二维游戏中通用的sprite（精灵）概念，主要是用来处理游

戏中的图像与动画的显示，这些都会作为JupiterSprite的成员（后有详细的类说明），

在绘图方面，使用了CxImage库（一个开源的图像操作类库，非常感谢作者Davide

Pizzolato），部分替代了windows中GDI+，使得gif动画更容易的绘制。在将游戏

中出现的所有物体都看做一个一个的精灵以后该怎么办？没错，再来一个管理的，这

就是JupiterSpriteManage，它将游戏中所有出现的精灵放在一起，能帮你查询，删

除你要的精灵，同时也统一了游戏的绘图。精灵准备就绪，想让它动起来，这时就需

要JupiterPhysics来帮你实现了，它实现精灵的运动，物理碰撞，真正的让你的精灵

动起来（i like to move it， move it！）。当然没有音乐怎么能行呢？JupiterMusic帮

你解决，虽然没有复杂的控制，但简单的播放暂停还是有的。终于，一切准备就绪，

Lets’s move it！

1. 游戏流程

新游戏

关于

退出

进入游戏

关于

退出

菜单

新游戏

处理

游戏结束

游戏循环

退出游戏

1. 代码规范
2. 所有头文件都使用#define防止被多重包括。格式为\_\_<FILE>\_，例如：

#ifndef \_\_EXAMPLE\_H\_\_

#define \_\_EXAMPLE\_H\_\_

...

#endif // \_\_EXAMPLE\_H\_\_

1. 使用前置声明，净量减少.h文件中的#include数量。例如，头文件中用到类File

但不需要访问File的定义，则头文件中需前置声明class File；而无需

#include “file.h”

1. 只有当函数只有10行或更少时才会将其定义为内联函数。
2. 参数顺序：定义函数时，输入参数在前，输出参数在后。
3. 仅当只有数据时使用struct，其他一律使用class。
4. 存取控制，将数据成员私有化，并提供相应的存取函数。
5. 声明次序，在类中使用特定的声明次序：public在private之前，成员函数在数据

变量之前。次序如下：

Public：， protected：， private： 如果没有，就忽略。在每一块中，声明次序如下：

1. typedef和enums
2. 常量
3. 构造函数
4. 析构函数
5. 成员函数
6. 数据成员
7. 编写短小的函数。
8. 每个文件加入版权公告，然后是文件描述符。例如：

/\*

\* Copyright(c), 2014 Jupiter-org

\* License Boilerplate: MIT

\*

\* @file: <file\_name>

\* @author: <author\_name>

\* @date:

\* @description:

\* @history: <author><time><version><desc>

\*/

1. 行长度不超过80（78）
2. 只是用空格，每行缩进4个空格。
3. 函数声明与定义。返回类型和函数名在同一行，合适的话，参数也放在同一行。

看起来像这样：

returnType className：：functionName（type param1， type param2）

{

doSomething();

...

}

如果同一行容不下参数：

returnType className：：reallyLongFunctionName（type param1,

type param2,

type param3)

{

doSomething();

...

}

注意以下几点：

1) 返回值总是和函数名在同一行；

2) 左圆括号（open parenthesis）总是和函数名在同一行；

3) 函数名和左圆括号间没有空格；

4) 圆括号与参数间没有空格；

5) 左大括号（open curly brace）总在最后一个参数下一行；

6) 右大括号（close curly brace）总是单独位于函数最后一行；

7) 右圆括号（close parenthesis）和左大括号间总是有一个空格；

8) 函数声明和实现处的所有形参名称必须保持一致；

9) 所有形参应尽可能对齐；

10) 缺省缩进为4个空格；

1. 函数调用尽量放在同一行，否则，将实参封装在圆括号中。调用遵循如下形式：

bool retval = doSomething(argument1, argument2, argument3);

如果同一行放不下，可断为多行，后面每一行都和第一个实参对齐，左圆括号后和右圆括号前不要留空格：

bool retval = doSomething(averyveryveryverylongargument1,

argument2, argument3);

如果函数参数比较多，可以出于可读性的考虑每行只放一个参数：

bool retval = doSomething(argument1,

argument2,

argument3,

argument4);

如果函数名太长，以至于超过行最大长度，可以将所有参数独立成行：

if (...) {

doSomethingThatRequiresALongFunctionName(

veryLongArgument1, // 4 space indent

argument2,

argument3,

argument4);

}

1. 条件语句：

if (condition) {

...

} else {

...

}

1. 循环和条件开关

while (condition) {

...

}

switch (var) {

case 0: {

...

break;

}

case 1: {

...

break;

}

default: {

...

}

}

1. 命名规则。采用小驼峰命名法。

函数名以小写开头，之后的单词首字母大写。

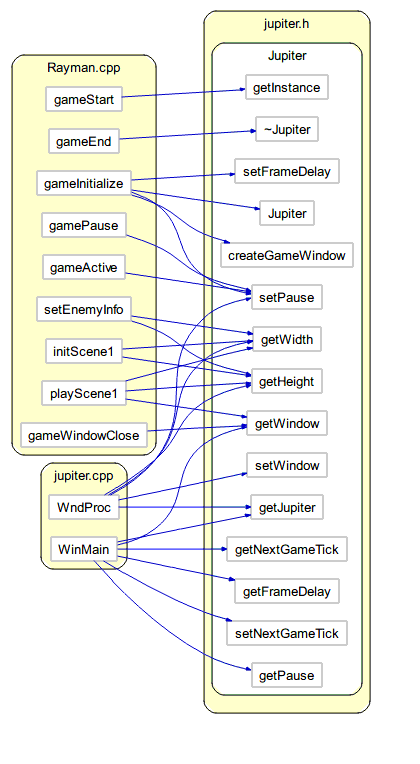
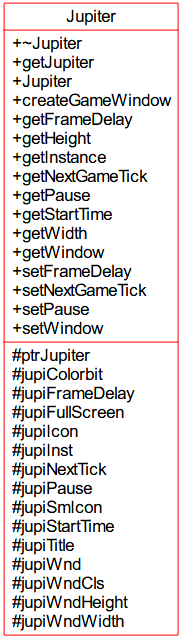
变量名义小写开头，之后的单词首字母大写。

常量以小写字母k开头，之后的单词首字母大写。

类型名一大写字母开头，之后的单词首字母大写。

1. 程序说明
2. 类的详细设计
3. Jupiter

用来在Win32环境下创建窗口，包含所有窗口信息，同时设置游戏的帧频。Rayman与Jupiter的调用关系如下：



游戏时间根据jupiNextTick和jupiFrameDelay来计算：

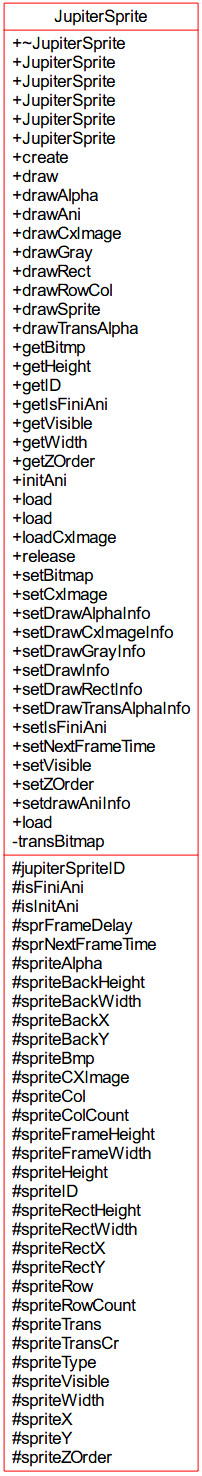
Jupiter中的jupiFrameDelay是帧频的倒数，以毫秒为单位，当游戏开始时，

程序会调用timeGetTime（）来获得当前系统时间保存在nTick里，然后根据

jupiFramDelay计算得到jupiNextTick，之后程序会不断地获得系统时间，并

和jupiNextTick进行比较，当jupiNextTick <= nTick，jupiNextTick就会被更新，只有当nTick > jupiNextTick时，游戏才会被渲染一次。

1. JupiterSprite



JupiterSprite中有一个静态变量jupiterSpriteID，每当使用JupiterSprite创

见对象时，该变量都会自增，同时会将其复制给spriteID，用作每个sprite的

标示符，在JupiterSpriteManage中可通过该标示符来查询、删除sprite。

每个精灵都有一个bmp对象和CXImage对象，CXImage处理除了位图以外的其他图像，用来弥补GDI+在绘制其他类型图像时的不足。为了统一精灵的

绘制，设定了精灵的绘制类型，声明为DRAYTYPE，每一种绘制在类型都对应

一个绘图方法。drawSprite(...)通过判断精灵的绘制类型来绘制。

DT\_DRAW对应最简单的绘图draw（...）

DT\_ALPHA根据设置的alpha来绘图drawAlpha(...)

DT\_GRAY将sprite的图片绘制为灰度图drawGray(...)

DT\_TRANSALPHA 去除精灵的背景drawTransAlpha(...)

DT\_ANIMA 如果精灵的图像是多张组成动画的图则使用drawAni(...)，其内部

调用了drawRowCol(...)

DT\_RECT drawRect(...)绘制图像上的一部分，由矩形来决定

DT\_CXIMAGE绘制除位图的其他图片drawCxImage(...)，主要是gif和png。

在进行绘制前需要设置绘图信息，每种绘图方式对应了一个设置方法，只要是

设置绘图方法，位置，透明度。内部不会调用，由类外部调用。

另外重要的一点是Z order，也就是精灵所处的层，在JupiterSpriteManage

中按Z order存储图像，会只是就会有层的效果。

主要属性：

spriteX， spriteY —— 绘制精灵的位置。

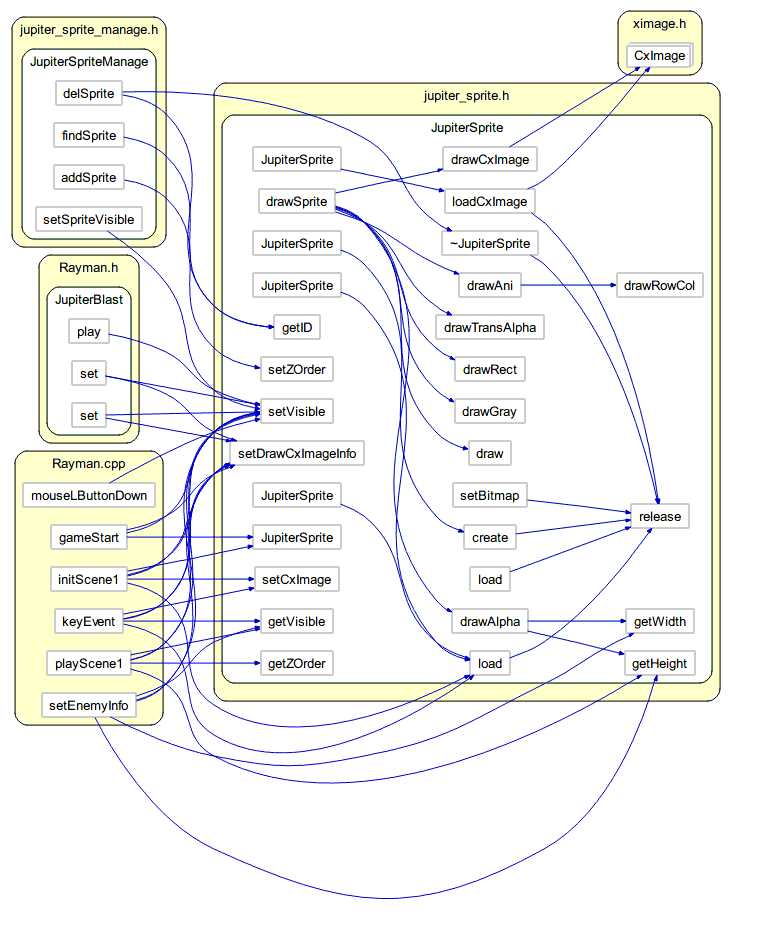
spriteTrans —— 精灵是否透明

spriteTransCr —— 精灵图当前的背景颜色，用来去除精灵的背景，制作镂空。

spriteAlpha —— 精灵的不透明度，影响到精灵的绘制。

spriteVisible —— 当此为false时，精灵不会被绘制，用于场景中精灵的管理。

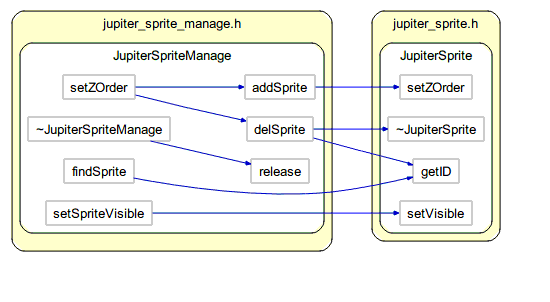
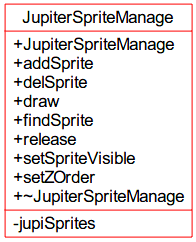
下图为其他类与JupiterSprite函数间的关系：



1. JupiterSpriteManage

JupiterSpriteManage使用STL中的vector来存储每个精灵，绘制时遍历所有精

灵，根据精灵的信息来绘制精灵。



1. JupiterPhysics

游戏中用来处理所有物体之间的碰撞。使用矩形框来做碰撞检测，为了避免出现

因为精灵图片大小比精灵本身大而出现误碰撞，碰撞框被设置成了精灵大小的90%，

可以有效避免错误。在游戏中，真正运行的是物理系统，他设置了每个物体的碰撞

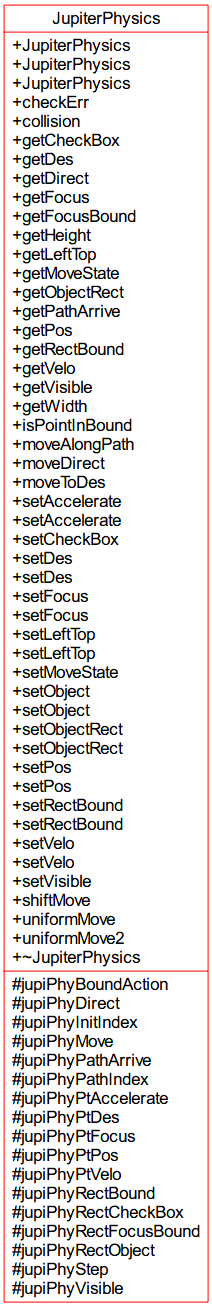
属性，并对用户输入做判断，只是在固定的时间画出图像。目前，JupiterPhysics

和JupiterSprite的绑定并不是很好，Rayman中，敌机只是通过绑定框的移动来

计算物体位置并传给精灵让其画图。

速度用一个二维点来表示，真正的速度由勾股定理算出。同时加入了按路径前进

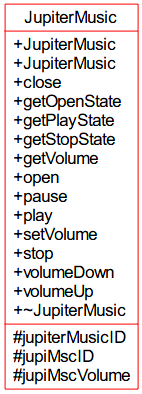
的功能，由数组来保存路径点，调用moveAlongPath()则会将精灵带到目的地。



1. JupiterMusic

音乐的播放使用了windows的播放多媒体文件的API指令函数，mciSendString。

能够播放MPEG，AVI，WAV，MP3类型的文件。



虽然是古老的代码，但仍能应付目前大多数音频。

本游戏用到3段音频，分别是menu页的欢迎BGM，战斗中BGM以及机体移动时的引擎轰鸣音效。

声明的函数虽有15个之多，但实际上在主程序调用时基本只用到play(), stop(), 以及调节音量的volumeDown()和volumeUp(),其余在申请空间时自动调用。

关于核心的命令串格式（因为每个函数基本都有，就以play（）为例）：

char szCommand[50]; ——缓冲区

sprintf(szCommand, "open %s ALIAS MUSIC%d", szMusicPath, jupiMscID); （这段其实就是将音频文件复制进缓冲区，open是操作，ALIAS MUSIC是别名，%d为ID）

if (!mciSendString(szCommand, NULL, 0, 0)) { // Open music file.

return true; }（操作正式进行，如注释所说！）

1. 游戏程序函数关系图

