GameMonkey Script FAQ

译者: ring.of.the.c@gmail.com
http://shaohui.me

Thanks for Matt, Greg and GameMonkey:)

什么是 GameMonkey?

GameMonkey 是一本专门为游戏和工具应用程序设计的脚本语言, 当然它亦适用于任何需要一个简单脚本环境支持的项目.GameMonkey 常被简称和缩写为 gm.

GameMonkey 的灵感和其中的基本概念来自于 lua, 但是从语法角度讲, 它更接近于 c. 这一特点使得它可以更容易被游戏编程人员接受. GameMonkey 原生提供线程机制和 states 的概念.

GameMonkey 是在 MIT 许可证下发行的, 这是一个比较”松”的许可证, 它唯一的要求就是, 你必须让别人知道它是 free 的. 你可以免费使用, 修改, 发行, 出售, 详细的权利和义务请参阅 MIT 许可证.

GameMonkey 的特色是什么?

1. 很小的代码基. 库大小只有 50kb 的内存占用(内存的占用受到共享库, 共享程序代码, 编译选项, 是否运行时编译的影响)
2. 可以运行时编译 GameMonkey 代码, 也可以链接那些预编译过的库.
3. 轻量级本地线程和状态机.
4. 软实时增量式垃圾回收机制. 标记可回收的内存, 不使用引用计数.
5. 方便 c\c++函数导入和调用, 亦可以从 c\c++中调用 GameMonkey.
6. 运行时调试和反馈的支持.
7. c 风格语法.
8. 在 cpu 占用和内存占用 在脚本语言中是具有竞争力的. 弹性, 简洁和速度的权衡.
9. GameMonkey 本身使用 flex, bison, c++编写, 在必要时易于修改.

GameMonkey 不是什么?

1. 注意, GameMonkey 不是为非程序员设计的, 也就是说, GameMonkey for programmer. 当然非程序员亦可以使用程序员提供的一组简洁的功能集去配置应用程序和定义一些简单的脚本行为.
2. GameMonkey 不适用于那些处理人命关天的任务. 比如远程手术等, 还

好我们游戏程序员不是处理人命关天的任务的人:)

GameMonkey 的跨平台性如何?

因为 GameMonkey 是用 c++编写的, 所以它可以运行在诸多平台上, 什么 win, linux, apple mac, xbox, sp2/3, psp, wii 等.

GameMonkey 从何而来?

在 2002 的第一场雪, matt 和 greg 尝试找到一种”做正确的事”的脚本语言, 我们认为的”正确的事”是指用可以通过这门脚本语言来提高游戏和工具应用程序的开发效率. 于是我们从文章, 总结, 讨论, 用户建议, 学生意见中寻找, 从 java 语言的进化过程, 一直到 c, python, 以及一大捆的在多种游戏中使用的脚本语言, 以及他们的源代码中比较和研究. 期间, 有一门名叫 lua 的语言逐渐引起了我们的重视. matt 在使用 lua 来粘合他自己的工具时获得了相当愉悦的使用体验. 我们认为 lua 就是我们想要的东西, 但是在 LuaV4(当时的最新版本), 我们认为还有很多想要的东西它并没有提供, 我们必须自己去实现.

比如说线程, 强壮的分析器, states 的概念, 像 c 一样的语法和基于 0 开始的习惯. 所以, 没有时间了, 我们必须拥有一门基于 lua 概念的脚本语言. 在那以后, 一个新的 console 项目开始了, 它就是 GameMonkey, 它的目标就是创建我们想象中的脚本语言来用于游戏和工具应用程序中. 同样, 这也意味着我们是在工作时间中来完成它的, 它将会成为公司私有的代码. 不幸的是, 几个月过后, 这个项目被取消了(在游戏行业里这很正常), matt 离开了这里去了别的地方工作. 我们请求将代码发布到社区, 让其他人受益, 并且利用社区力量来继续推动它的开发. 在 2003-8-12, 公司终于同意了我们的请求, 把 GameMonkey 的源代码在我们挑选的 free software 和 open source 许可授权下发布了, 我们不能出售源代码, 或者从中受益. 但我们要感谢 auran 公司!

为什么它叫做 GameMonkey?

我们想要一个独一无二的名字, 这个名字必须能够体现 GameMonkey 是面向游戏编程, 自然的, 响亮的, 而且必须很酷等等. 在挑选之后, 我们决定使用 GameMonkey 作为它的名字. 选这个名字可能有其他下意识…或者潜意识的的原因, 但是我们喜欢它.

从 GameMonkey 的最新版本中, 我能获得什么?

1. GameMonkey 语言和一些相关的构件.
2. 在命令行中运行 GameMonkey Script 的例子.

3. lib.
4. 一个演示单步，查看变量和断点的调试器的例子.
5. 一些 GameMonkey Script.

GameMonkey 在下一阶段要引入的特性?

1. 新的寄存器堆栈虚拟机，将提升效率到现在虚拟机的二倍.
2. 内置调试器的 ide.
3. 更多的有用的工具和应用.
4. 原生支持枚举和常量.

我能给 GameMonkey 提供什么帮助?

1. 在你的项目中使用 GameMonkey.
2. 为 GameMonkey 制作一些工具并分享它.
3. 向别人介绍 GameMonkey.
4. 改进增强 GameMonkey 的调试器，文档，例子，等等.
5. 还有很多可以做的事情，比如提供 GameMonkey 的下载镜像

第三方工具

scintilla 用于制作调试器. http://www.scintilla.org
Flex 和 Bison 用于编译器 http://www.gnu.org/software/flex 和 http://www.gnu.org/software/bison/bison .

在下载的 GameMonkey 包中都有什么?

```
gmsrc/bin                          编译器执行文件
        /doc                          文档
        /EditorHighlighters   语法高亮的编辑器
        /scripts                      脚本例子
        /src/binds                 gm 的绑定例子
              /examples           程序例子
              /gm                    GameMonkey 脚本源码
              /gmd                  调试器例子
              /gme                  脚本执行器例子
              /gml                   GameMonkey 的 lib
              /platform            平台相关头文件
```

源码目录下有 VS6 的项目文件和 workspace 文件.

用 GameMonkey 编写的代码是什么样子的?

1. 下面是一些用 GameMonkey 编写的代码:

```
OnDoorTriggerEnter = function(door, objEntering)
{
    if (objEntering == player && !door.IsOpen())  // 进入的对象是一个玩家, 而且门当前是锁着的
    {
        door.Open();  // 开门
        return true;
    }
    return false;
}
```

2. 下面再展示如何从把 c 函数导入到 GameMonkey 中

```
#include “gmThread.h”
#include “gmMachine.h”

// float SquareRoot(int / float)
int GM_CDECL SquareRoot(gmThread* a_thread)
// 注意, 要导入到 GameMonkey 中的 c 函数的原型必须是  int GM_CDECL funcname(gmThread* thread)
{
    float fValue;
    GM_CHECK_NUM_PARAMS(1);  //检查, 有一个参数

    // 根据参数的类型, 获取参数
    if (a_thread->ParamType(0) == GM_INT) {
        fValue = (float) a_thread->Param(0).m_value.m_int;
    }
    else if (a_thread->ParamType(0) == GM_FLOAT) {
        fValue = a_thread->Param(0).m_value.m_float;
    }
    else {
        return GM_EXCEPTION;
    }

    // 处理数据并返回
    a_thread->PushFloat(sqrtf(fValue));
    return GM_OK;
}
```

注意下面是怎么将 SquareRoot 这个 c 函数导入到 GameMonkey 中

```
extern gmMachine* machine;
machine->RegisterLibraryFunction(“SquareRoot”, SquareRoot);
```

3. 下面演示 c 中如何调用 GameMonkey 中的函数

```
#include “gmMachine.h”
#include “gmCall.h”
extern gmMachine machine;
// 假设在 GameMonkey 中存在 int Add(int. int) 这样一个函数
int AddTwoIntegers(int valueA, int valueB)
{
      int resultInt = 0;
      gmCall call;
      if (call.BeginGlobalFunction(&machine, “Add”)) {
           call.AddParamInt(valueA); // 压入第一个参数
           call.AddParamInt(valueB); // 压入第二个参数
           call.End();                          // 调用
           call.GetReturnedInt(resultInt); // 取返回值
      }
      return resultInt;
}
```

如何运行第一个 GameMonkey 例子?

按照传统的惯例, 我们应该先演示 hello world 小例子:)

1) 先得在你的硬盘上解压下到的 GameMonkey 压缩包
2) 创建一个 test.gm 文件
3) 键入’ print(“Hello World”); ‘
4) 将 test.gm 保存到 …../gmsrc/bin 下
5) 在命令行中键入 ‘ gme test.gm ‘

GameMonkey Script Executer 的使用说明

gme.exe 只是一个简单的基于命令行的 GameMonkey 脚本解释程序, 就想 lua.exe 一样. 你可以使用它来编写 console game 和 tool, 或者是一些演示说明脚本. gme.exe 只是一个演示工具, GameMonkey 真正的威力显示在当你在自己

的程序中使用它来做扩展，配置和逻辑的时候. 现在，你可能已经想要使用gme来体验这门语言或者是用它来执行tool 脚本了，就想.bat和.com文件一样，gme也有自己的使用语法:

gme.exe <script file> <gme commands> <script file commands>
<script file> — 将要执行的脚本文件，通常是.gm文件
<gme commands> — 是由以下的组合:
-k 运行结束后按键才推出，方便看运行结果
-d 允许调试器的接入
-e 在全局表’env’中加上windows 的环境变量
-ke 在发生编译和run-time错误时按键才推出，方便看错误信息
<scirpt file commands> — 任何你想传递给脚本的参数，这些参数将被存在一个全局表’arg’里面

第一个嵌入式GameMonkey虚拟机

这才是真正的威力所在:)在你自己的应用中嵌入GameMonkey很简单，就是将GameMonkey的源代码和你的应用的源代码一起编译，然后创建一个gmMachine，并用它执行gm脚本即可. 下面演示一下简单的应用:

为了嵌入式GameMonkey，你通常需要一下步骤:

1) 将src/gm下除了gmDebugger[只有需要创建特定的调试器时才使用gmDebugger] 的所有 .cpp和 .h文件加入到你的项目中.

2) 加入平台依赖文件，比如win32就是src/platform/win32/gmConfig_p.h.

3) 加入你需要的辅助工具源文件[如binding等]，在src/binds中.

4) 编译，可以把他们和你的应用混在一起编译，亦可以把GameMonkey单独编译成一个库，这样的话，你需要在你的应用中引入lib和配置lib的位置.

5) 创建你自己应用的binds,使用src/binds文件中的例子,也可以参考FAQ和GM参考手册上的例子.

下面是win32下的一个console的例子.

1. 最小的例子

```
#include “gmThread.h”    // GameMonkey Script
int main(int argc, char* argv[]) {
    // 1. 创建GM虚拟机
    gmMachine machine;
    // 2. 编译并执行GM语句
    machine.ExecuteString(“print( ‘Hello world’ );”);
    getchar();
```

```
    return 0;
}
```

2. 可以交互的例子

```
#include <windows.h>
#include <mmsystem.h>
#include “gmThread.h”
int main(int argc, char*argv[]) {
//1. 创建 GM 虚拟机
gmMachine* machine = new gmMachine;

// 2. 从 stdin 中获取 GM 脚本
fprintf(stdout, “Please enter one line of script\n>”);
const int MAX_SCRIPT_SIZE = 4096;
char script[MAX_SCRIPT_SIZE];
fgets(script, MAX_SCRIPT_SIZE - 1, stdin);

// 3. 编译, 但不马上执行 GM 脚本
int errors = machine->ExecuteString(script, NULL, false, NULL);

// 4. 验错
if (errors){
    bool first = true;
    const char* message;
    while ( (message = machine->GetLog().GetEntry(first) ) )
    fprintf(stderr, “%s” GM_NL, message);
    goto _Exit;
}

// 5. 运行
int deltaTime = 0;
int lastTime = timeGetTime();
while (machine->Execute(deltaTime)){
    int curTime = timeGetTime();
    deltaTime = curTime - lastTime;
    lastTime = curTime;
}

// 6. 清理
_Exit:
    delete machine;
    getchar();
```

```
return 0;
}
```