从零开始实现GameBoy模拟器 #0 开篇

銀葉吉祥
浙江大学 软件工程硕士

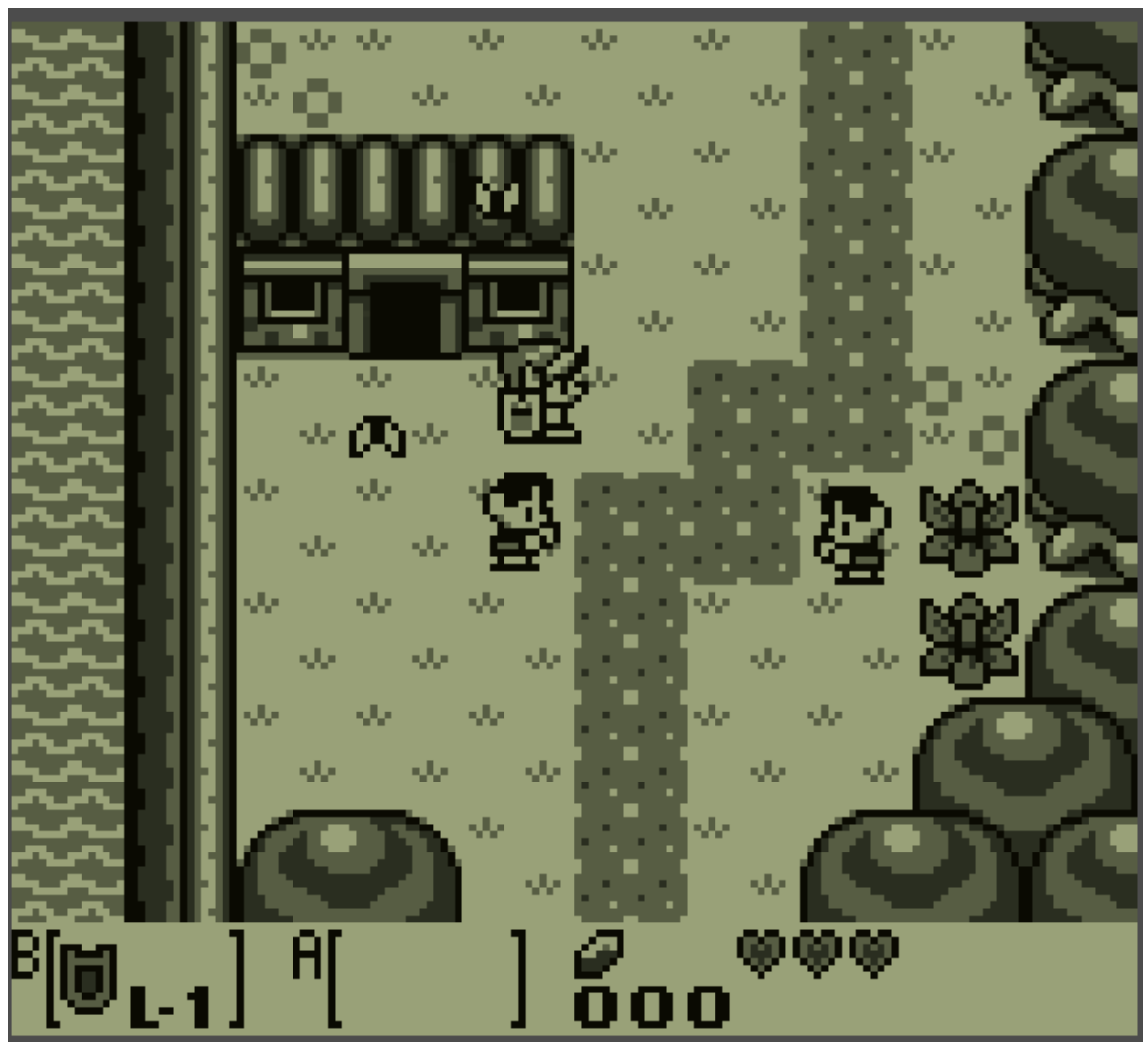
本专题系列文章将手把手带领你从零开始用C++实现一个GameBoy模拟器。文章将会首先介绍GameBoy的硬件架构，然后将GameBoy拆散，一步步带你实现GameBoy的CPU、总线、PPU、APU等组件，并最终组装成一个全功能的GameBoy模拟器，你可以使用该模拟器加载任何GameBoy的ROM卡带文件，并在Windows和macOS上畅玩GameBoy游戏。本专题由十六篇文章构成，每篇文章包含了大约一天的学习量，预计读者需要花费半个月左右的时间才能完成所有功能的设计和调试。本专题提供了配套的项目代码仓库，每一篇专题文章都在代码仓库中有自己的独立目录，供读者参考实现，我们建议读者在阅读文章时，先通读一遍文章，然后隐藏文章开始自己实现其中描述的组件，最后在出现问题时对照着仓库代码来检查自己的实现。GameBoy模拟器的开发并非易事，相信在旅程的终点，你能够有所收获。

为什么要实现一个GameBoy模拟器

在学习一门技术时，效率最高的学习方法之一就是找一个难度适中，具有典型性的案例，并自己实现案例中的成果，无论其是一件真实的工艺品、艺术品，还是一个影视剪辑片段，一个图书馆管理系统，亦或是本文中介绍的GameBoy模拟器。作者认为，实现一个GameBoy模拟器对于读者编程能力的提升和对计算机体系的理解具有非常大的帮助。首先，虽然现代的计算机在形式上多种多样，但它们在本质上其实大同小异，都由中央处理器（CPU）、只读存储器（ROM）、可读写存储器（RAM）和各种外围设备（显示屏、键盘、扬声器等）构成，而这些组件在一台GameBoy上都有体现。通过实现一个GameBoy模拟器，读者可以对一个现代计算机如何工作有深刻的理解，这些经验可以帮助读者在未来编写虚拟机、硬件驱动、脚本语言运行时等底层框架时更加游刃有余。其次，GameBoy模拟器的编写过程十分有趣。作为全球销量排名前五的游戏机，GameBoy具有非常多的经典作品，例如《俄罗斯方块》、《网球》、《塞尔达传说：织梦岛》、《恶魔城：德古拉传说》等，其代表的8bit游戏机和游戏已经深入如今的游戏文化中。在我们开发GameBoy模拟器的过程中，我们会使用真正的GameBoy游戏卡带ROM文件来测试我们的模拟器，看着这些经典游戏真正运行在自己编写的代码上，想必读者一定会感到满满的成就感。

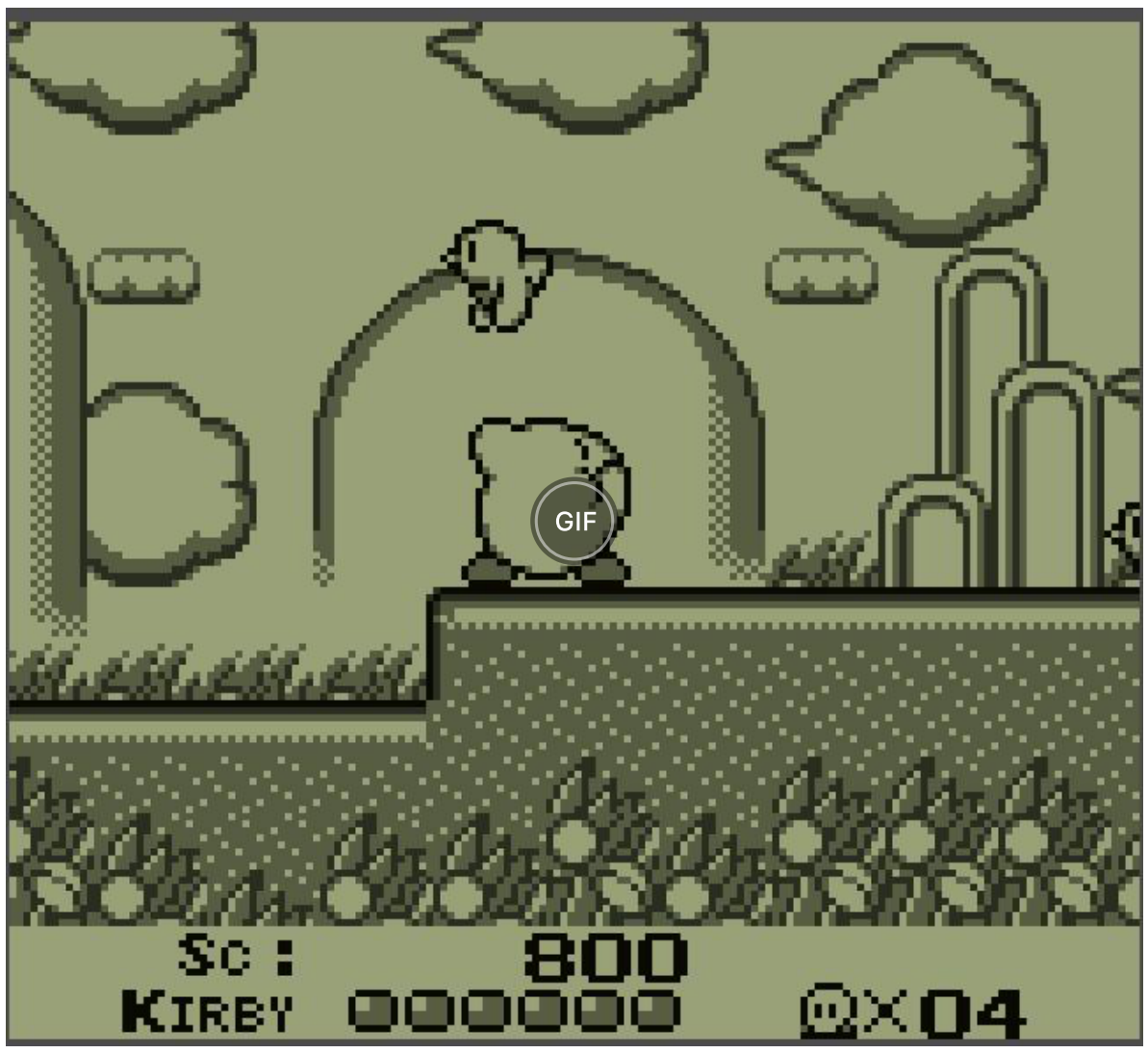
总之，请享受这一段GameBoy模拟器的开发之旅吧。

一些效果展示



塞尔达传说：织梦岛

恶魔城：德古拉传说



星之卡比

一些参考资料

由于GameBoy诞生时间早，影响范围大，对于GameBoy的各个方面的研究已经相当详细，这些资料目前都可以在网上查到，作者总结了其中一些比较全面的资料供读者参考。在之后的开发过程中，作者也会不时引证这些资料中的信息，以方便读者理解GameBoy的一些概念。

Game Boy的硬件剖析
www.bilibili.com/video/BV1QA4y1o7qx

这个视频详细讲解了GameBoy的硬件原理，包括其CPU如何工作的，CPU如何与各种外设通信的，诸如此类，可惜视频只出了三集就断更了，但是依然是非常不错的学习资料。

Gameboy Emulator Development
www.youtube.com/watch?v=e87qKixKFME&list=PLVxiWM

这个视频教程系列手把手教你实现GameBoy模拟器的大部分功能，也是作者在第一次尝试编写GameBoy模拟器时的主要参考资料。这个视频教程分16集，每一集在半小时左右，内容详实，节奏合理，但是也断更了，没有实现GameBoy的APU（音频处理单元）部分。相比这个视频教程专题，咱们的专题会补全GameBoy的APU部分，并修复其中的一些作者在跟着做时发现的bug，因此跟着这篇文章学习就行了 XD。

Foreword - Pan Docs
gbdev.io/pandocs/

该文档自称为你能在公共互联网上找到的最全面的GameBoy文档。Well, almost. 除了下文提到的GameBoy任天堂官方文档以外，这确实是你能找到最详细的GameBoy文档了。事实上，在某些方面，该文档的描述甚至比任天堂官方手册更加详细，因为其中的很大一部分内容来自于GameBoy社区对GameBoy硬件的实际数据抓取，所以有一些并没有出现在任天堂官方文档中的特性也会在该文档中有所提及。本专题中大部分对于硬件行为和寄存器的描述都可以在该文档中找到对应的参考，因此笔者强烈建议在实现的时候参考该文档，以学习GameBoy硬件的各种特性。

https://archive.org/details/GameBoyProgManVer1.1/page/n85/mode/2up
archive.org/details/GameBoyProgManVer1.1/page/n85/mo

该文档是任天堂官方的GameBoy编程手册，主要面向基于GameBoy开发游戏的厂商，以提供GameBoy的全面编程参考。该文档详细规定了GameBoy的硬件行为，因此在实现模拟器时具有非常大的参考价值。

Gameboy (LR35902) OPCODES
www.pastraiser.com/cpu/gameboy/gameboy_opcodes.ht

该网页详细展示了Gameboy的CPU（LR35902）的指令集，包括每一个指令集的字节长度，所需要的时钟周期，以及其会影响的标志位等信息。在进行CPU指令集实现时，我们将会频繁用到该网页中所述的内容。

https://9ioldgame.com/games/Nintendo-GAMEBOY-Games.html
9ioldgame.com/games/Nintendo-GAMEBOY-Games.html

此处可以下载GameBoy的游戏资源，以便测试我们的模拟器使用。

启航前的准备

在我们正式开始GameBoy模拟器开发之旅前，我们需要做一些准备工作。

首先，读者可以加入本专题的交流群：338547343，一起讨论本专题相关的技术内容。

然后，我们需要安装这个专题系列所需的开发环境：

1. 对于Windows系统，需要下载Visual Studio 2019及更新的版本，并在安装器中选择“使用C++的桌面开发”和“使用C++的游戏开发”这两项工作负载。
2. 对于macOS系统，需要在App Store中下载并安装XCode。在安装完XCode以后，需要从启动台中启动XCode。首次启动XCode时会下载和安装XCode的各种命令行工具，请务必等到所有工具都下载完毕以后再关闭XCode。
3. 下载并安装xmake，这是本项目使用的构建系统和包管理系统。可以参考这篇文章安装xmake。对于Windows系统，可以在PowerShell中运行以下脚本安装：

```
Invoke-Expression (Invoke-Webrequest 'https://xmake.io/psget.text' -UseBasicPa
```

对于macOS，则可以在终端中运行以下脚本安装：

```
curl -fsSL https://xmake.io/shget.text | bash
```

安装完毕后，可以尝试运行以下命令确保xmake正确安装：

```
xmake --version
```

这就是我们需要配置的全部环境了！所有其余依赖项都会在我们编译工程时由xmake自动下载，开箱即用。

接下来，我们需要下载项目使用的工程文件，地址如下：

JX-Master/LunaGB - GitHub
github.com/JX-Master/LunaGB.git

您可以选择使用git clone项目，或者直接从GitHub下载zip文件，两种方式都可以正常完成后续的步骤。作者建议使用git clone来下载项目，这样之后的项目代码更新可以直接通过git pull的方式同步，不用重复下载和覆盖。

在下载项目至本地后，我们需要先点击setup.bat（mac用户则需要打开终端，运行setup.sh）。该脚本会自动下载本项目依赖的SDK，并放在项目目录的SDKs文件夹下。

在SDK下载完毕后，我们就可以开始尝试编译项目了：

1. 【推荐的方式】如果您使用Visual Studio Code或者命令行，可以在项目根目录中打开VSCode或者命令行。
2. 如果您使用Visual Studio，可以运行项目根目录下的gen_vs2019.bat脚本，xmake会自动在Solution目录下生成Visual Studio的解决方案。
3. 如果您使用XCode，可以运行项目根目录下的gen_xcode.sh脚本，xmake会自动在Solution目录下生成XCode的工程文件。

如果您使用Visual Studio Code或者命令行，则可以使用以下命令配置和构建项目：

```
xmake build
```

在第一次运行时，xmake会提示需要下载安装一些依赖库，在安装完成后，您应该得到类似如下

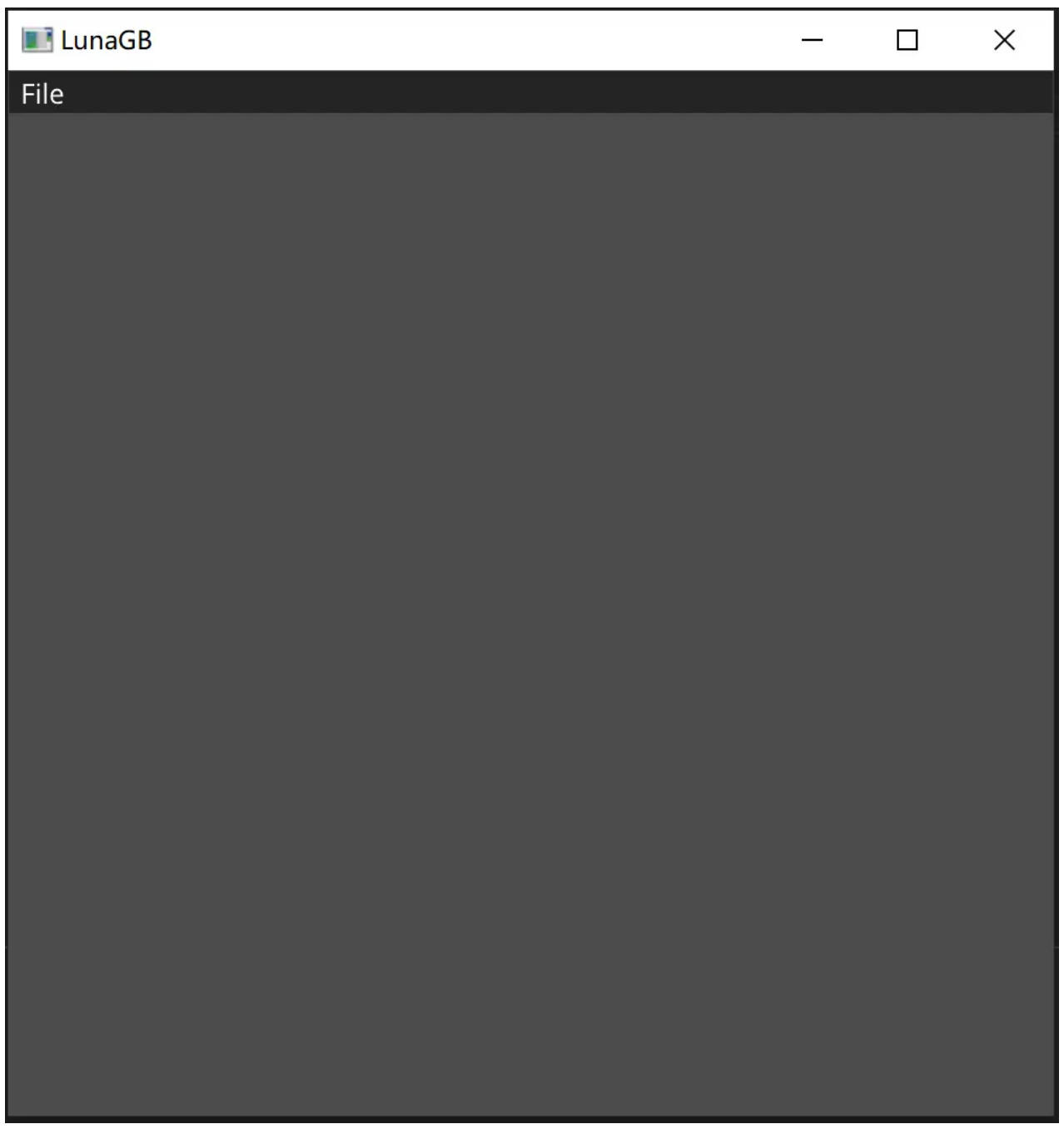
输出：

```
[ 79%]: archiving.release LunaVFS.lib
[ 79%]: archiving.release LunaJobSystem.lib
[ 79%]: archiving.release LunaWindow.lib
[ 79%]: archiving.release LunaHID.lib
[ 79%]: archiving.release LunaFont.lib
[ 79%]: archiving.release LunaAHI.lib
[ 79%]: archiving.release LunaNetwork.lib
[ 91%]: archiving.release LunaShaderCompiler.lib
[ 91%]: archiving.release LunaAsset.lib
[ 91%]: archiving.release LunaECS.lib
[ 91%]: archiving.release LunaRHI.lib
[ 95%]: archiving.release LunaVG.lib
[ 95%]: archiving.release LunaImGui.lib
[ 95%]: archiving.release LunaRG.lib
[ 98%]: linking.release LunaGB.exe
[100%]: build ok, spent 6.235s
```

这表明您已经能够正常编译项目。接下来执行以下命令：

```
xmake run LunaGB
```

您应该可以看到一个窗口，这就是我们的模拟器的图形界面：



如果看到了这个窗口，那么恭喜你，你已经完成了启航前的准备，可以正式开始GameBoy模拟器的开发之旅了！

项目结构

作为GameBoy模拟器开发的第一站，就让我们先熟悉一下我们模拟器项目的文件结构，以及模拟器初始代码的含义吧！

我们的模拟器使用LunaSDK实现，这是一个笔者自己开发的软件开发套件，可以用于开发跨平台的图形程序。如果读者在之前的编程中使用过SDL、openFrameworks等框架，那么对LunaSDK的各项功能应该可以快速上手；如果读者并没有这方面的经验，也没关系，在本项目中，我们只会使用LunaSDK的基础功能实现一些必需的交互（例如展示游戏画面、接收输入、播放音频等），并且对于每一个用到的功能，笔者都会进行讲解，保证读者能够顺利完成模拟器的开发。

同时，对于想要详细了解LunaSDK的读者，也可以参考GitHub的页面和文档站：

JX-Master/LunaSDK - GitHub
github.com/JX-Master/LunaSDK.git

LunaSDK Docs
www.lunasdk.org/

本项目的文件结构采用LunaSDK的项目结构，根目录下主要包括三个文件夹：

1. Modules：存放LunaSDK的模块代码。
2. Programs：存放基于LunaSDK的程序的代码。
3. build：当我们使用xmake构建程序时，该目录用于存放构建出来的可执行文件和二进制库文件。

其中，我们需要重点关注的是Programs文件夹，该文件夹会存放我们模拟器项目的所有代码。其中，LunaGB文件夹存放模拟器程序的基础代码，即在上一步中运行的代码。读者需要跟着本专题教程，基于该基础代码逐步实现一个完整的模拟器。同时，LunaGB-XX为作者提供的参考代码，其中的XX代表了对应的专题文章序号（从01开始）。在读者跟着文章实现对应的功能时，可以随时查阅文章对应的参考代码，以检查自己的代码是否有写错的地方，并及时更正。

初始的LunaGB文件夹包括三个文件：

1. main.cpp：不用多说，这是我们的应用程序的入口。
2. App.hpp/App.cpp：定义了一个App类，用来存放我们应用程序的全局变量和程序主循环。

我们先从main.cpp的代码看起：

```
App* g_app;
int main()
{
    bool inited = Luna::init();
    if(!inited) return -1;
    g_app = memnew<App>();
    RV r = run_app();
    if(failed(r))
    {
        log_error("LunaGB", explain(r.errcode()));
    }
    memdelete(g_app);
    Luna::close();
    return 0;
}
```

应用程序运行以后，首先会调用Luna::init函数，该函数负责初始化LunaSDK的核心功能。与之对应的是Luna::close函数，其负责关闭LunaSDK库，并清理所有使用到的资源。Luna::init返回一个bool值来表示其是否初始化成功，如果其初始化不成功，则我们直接返回-1，提示系统程序遇到了错误。

在初始化LunaSDK以后，我们需要创建一个App对象来保存程序的全局状态。LunaSDK提供了memnew<T>和memdelete函数用于动态创建和销毁对象，读者可以将其理解为C++的new/delete的关键字在LunaSDK中的替代。在创建对象以后，我们调用了run_app来运行我们的程序：

```
RV run_app()
{
    lutry
    {
        // Add modules.
        luexp(add_modules({module_window(), module_rhi(), module_ahi(), module_
        // Initialize modules.
        luexp(init_modules());
        // Run the application.
        luexp(g_app->init());
        while(!g_app->is_exiting)
        {
            luexp(g_app->update());
        }
    }
    lucatchret;
    return ok;
}
```

首先需要注意的便是该函数的返回值：RV。RV实际上是R<void>的缩写，其中的R代表Result，表示该函数不返回任何值，但是可能**返回错误**。在LunaSDK中，我们使用R<T>模板封装任何可能返回错误的函数，这些函数可以选择在成功执行时按照正常的逻辑返回返回值，或者在发生错误时返回错误码。在main函数中，我们使用failed函数来检查我们的运行是否出错，并在产生错误时调用log_error输出错误信息。

在run_app中，我们首先调用了add_modules导入我们程序需要使用到的LunaSDK模块，然后调用init_modules初始化这些导入的模块，最后调用App::init初始化我们的应用程序。此处的问题在于，这三个函数都有可能出错并返回错误码，而为每一个函数都写一个if(failed(...))去检查这些错误，未免有些繁琐。因此，LunaSDK借用了C++异常处理的写法，使用一组宏来包覆可能出错的函数调用，从而简化代码。这些宏包括：

1. lutry { ... } 定义了一个尝试语句块，在语句块中的代码可能会产生错误。
2. lucatch { ... } 定义了一个捕获语句块，当lutry语句块中的代码产生错误时，执行流程将会跳转到lucatch语句块，且错误码可以使用luerr宏获取。
3. lucatchret相当于lucatch { return luerr; }，表示我们不想在当前函数中处理错误，而是将错误返回给调用方处理。如果需要返回错误码，则本函数的返回值必须使用R<T>模板包覆。
4. luexp(...)包覆了一个返回值为RV的函数调用，并在函数返回错误时自动跳转到lucatch语句块。
5. lulet(a, ...)包覆了一个返回值为R<T>的函数调用。当函数调用成功时，其定义一个类型为T的局部变量a接收函数的返回值；当函数调用错误时，其自动跳转到lucatch语句块。
6. luset(a, ...)与lulet(a, ...)功能类似，不过其是将返回值赋予到一个现有的变量a上，而非创建一个新的局部变量。

在了解了这些宏的写法以后，上面的代码就变得非常简单：其初始化了程序，然后无限循环更新程序，直到程序被用户退出。

接下来让我们继续看App.hpp的代码：

```
#pragma once
#include <Luna/RHI/Device.hpp>
using namespace Luna;

struct App
{
    //! `true` if the application is exiting. For example, if the user presses
    //! button of the window.
    bool is_exiting;

    //! The RHI device used to render interfaces.
    Ref<RHI::IDevice> rhi_device;
    //! The graphics queue index.
    u32 rhi_queue_index;

    //! The application main window.
    Ref<Window::IWindow> window;
    //! The application main window swap chain.
    Ref<RHI::ISwapChain> swap_chain;
    //! The command buffer used to submit draw calls.
    Ref<RHI::ICommandBuffer> cmdbuf;

    RV init();
    RV update();
    ~App();

    void draw_gui();
    void draw_main_menu_bar();
};
```

App类定义了创建一个图形应用程序所必需的对象，例如Window::IWindow表示一个系统窗口，RHI::IDevice表示用于绘制的图形设备，RHI::ISwapChain表示窗口缓存的交换链等。LunaSDK的大部分对象均通过带有引用计数的接口指针提供，Ref<T>智能指针用于自动维护这些对象的生命周期，因此我们无需手动管理它们。由于本专题的重点在于对模拟器的实现，因此对于这些图形应用程序相关的对象不会进行过多的说明。读者若是对相关内容感兴趣，可以参考LunaSDK的文档以了解更多内容。

App.cpp中定义了我们的程序的具体逻辑，部分次要代码已省略（使用/*...*/表示），以节省篇幅：

```
#include "App.hpp"
#include <Luna/Runtime/Log.hpp>
#include <Luna/ImGui/ImGui.hpp>

RV App::init()
{
```

```
    lutry
    {
        is_exiting = false;
        // In order to see LunaSDK logs.
        set_log_to_platform_enabled(true);
        // Create window, create RHI device, create swap chain, and so on...
        /*...*/
    }
    lucatchret;
    return ok;
}
RV App::update()
{
    lutry
    {
        // Update window events.
        Window::poll_events();
        // Exit the program if the window is closed.
        if (window->is_closed())
        {
            is_exiting = true;
            return ok;
        }

        // Draw GUI.
        draw_gui();

        // Clear back buffer.
        /*...*/
        // Render GUI.
        luexp(ImGuiUtils::render_draw_data(ImGui::GetDrawData(), cmdbuf, back_b

        // Submit render commands and present the back buffer.
        /*...*/
    }
    lucatchret;
    return ok;
}
App::~App()
{
    // TODO...
}
void App::draw_gui()
{
    // Begin GUI.
    ImGuiUtils::update_io();
    ImGui::NewFrame();
    draw_main_menu_bar();
    // End GUI.
    ImGui::Render();
}
void App::draw_main_menu_bar()
{
    if (ImGui::BeginMainMenuBar())
    {
        if (ImGui::BeginMenu("File"))
        {
            if(ImGui::MenuItem("Open"))
            {
                // TODO...
            }
            ImGui::EndMenu();
        }
        ImGui::EndMainMenuBar();
    }
}
```

在App::init中，我们主要创建了窗口应用程序所需的窗口、图形设备，交换链等对象，接着便进入我们程序的主循环：App::update。在App::update中，我们首先查询系统的窗口事件，并处理这些窗口事件，其中如果用户点击了窗口的关闭按钮，则我们将is_exiting设置为true，并中断主循环。在处理完事件以后，我们调用draw_gui绘制窗口的图形用户界面。LunaSDK使用Dear ImGui框架进行图形用户界面的绘制，目前我们唯一的图形用户界面是一个主菜单，包括File->Open一个选项，该选项会在下一章中用于加载GameBoy的卡带ROM文件。我们将在之后的章节中为程序添加更多的窗口和功能，方便我们对模拟器的运行状态进行跟踪和调试。
App::update的最后步骤是等待这一帧的内容绘制完成，并将这一帧中绘制的所有内容展示到窗口上，然后等待显示器的垂直同步。

以上就是本章节的全部内容。从下一章开始，我们将正式开始GameBoy模拟器的编程之旅。我们将会了解到GameBoy的硬件基本信息，并动手实现GameBoy卡带的加载。那么，下一章见！

銀葉吉祥：从零开始实现GameBoy模拟器 #1 加载卡带
106 赞同 · 7 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #2 CPU、时钟和总线
72 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #3 加载，比较和跳转指令
22 赞同 · 2 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #4 逻辑和运算指令
22 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #5 CB指令、特殊指令、中断
17 赞同 · 2 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #6 定时器、串口、调试面板
20 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #7 PPU
37 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #8 绘制背景和窗口
20 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #9 绘制精灵
19 赞同 · 2 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #10 按键输入、MBC1卡带
13 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #11 MBC2、MBC3卡带
14 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #12 APU
10 赞同 · 2 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #13 脉冲音频通道
12 赞同 · 0 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #14 波形和噪声音频通道
15 赞同 · 1 评论 文章

銀葉吉祥：从零开始实现GameBoy模拟器 #15 高通滤波器，总结
28 赞同 · 3 评论 文章

编辑于 2024-03-19 22:30 · IP 属地上海

「真诚赞赏，手留余香」

赞赏

1 人已赞赏

游戏机模拟器 Game Boy（GB）

362

欢迎参与讨论

23 条评论 默认 最新

小学生读金玉灯
😏这篇都发了几十分钟了，下一篇该写好了吧?
01-18 · IP 属地四川 回复 5

微笑
就喜欢这样带大家入门的
01-18 · IP 属地江苏 回复 2

Egnoty
《难度适中》、《具有典型性》
01-19 · IP 属地浙江 回复 2

梵天
You are God damn hero!
01-30 · IP 属地山西 回复 1

胡鸿飞
这么离谱的教程么？光看个简介就已经觉得是我不可能完成任务了……
01-18 · IP 属地江苏 回复 1

留川
笔记写得不错，就是项目dependence下不下来，挂了代理还是不行。手动下了，放到指定目录，结果编译又出问题，vs2019,vs2022都试了。反正就是特别烦的那种排查工程。要是作者把依赖库用submodule的方式放进repo里就好了。
04-08 · IP 属地上海 回复 喜欢

銀葉吉祥 作者
xmake的proxy得用xmake g —proxy="127.0.0.1:7890"单独配，不然是不会生效的🤦
04-10 · IP 属地上海 回复 喜欢

浅若夏沫
我在进行xbuild 过程中出现了编译错误，这是为什么呢
04-07 · IP 属地四川 回复 喜欢

臧大为
😳你这个太硬核了，连图形框架都是自研的！
03-16 · IP 属地中国香港 回复 喜欢

三鱼
而你，我的朋友，你是真正的英雄😚
03-13 · IP 属地江苏 回复 喜欢

葡萄园公告牌
啊?
02-14 · IP 属地湖南 回复 喜欢

点击查看全部评论 ›

362

欢迎参与讨论

文章被以下专栏收录

吉祥的游戏制作笔记
游戏制作中的技术、艺术和设计灵感记录

推荐阅读

那些稀奇古怪的“XX模拟器”为何开始大行其道了?

游研社 发表于游研社

一款游戏模拟器，上万款游戏随便耍

编辑 | 排版 | 制图 | 测试 | ©离城此文用时1小时23分钟，希望大家支持，每天坚持更新真的不易，希望我的每一份劳动成果都可以得到大家的一个【在看】，所有分享资源获取均在文末 资源君今...

软知否

任天堂手撕模拟器，但模拟器真的该死吗?

战术大米 发表于干杯，世界...

嘿嘿嘿，你一定玩过模拟器

火狼 发表于有趣点