SimpleGUI

HMI 模型简要说明

摘要

简单说明 HMI 引擎的作用与运行机理。

Polarix

Xuyulin91@163.com

1、 说明

SimpleGUI 的演示例程中涉及到了一个简单的用于界面交互的机制,我称之为 HMI (Human Machine Interface) 引擎,本文将对在这个 HMI 引擎做简单的原理阐述和使用说明。

首先声明,这个 HMI 引擎是一个临时的式作品,在 SimpleGUI 的不断改进过程中, HMI 引擎将可能发生结构性的变化,介时此文档将会有对应的更新,敬请关注。

2、 设计背景

HMI 引擎本身和 SimpleGUI 没有具体的联系,SimpleGUI 的本质是实现点、线、平面几何体、位图(包括文字)的绘制,进而将这些基本的绘图操作进行组合,实现诸如提示框(NoticeBox)、列表(List)等的显示反馈。进而用户可以使用这些绘图接口,构建自己需要的界面,而对于列表、文本框等,也提供了相关的操作做和更新接口。按照SimpleGUI 的原始设计来讲,每一个 SimpleGUI 提供的界面或控件,只要原始数据不改变,就可以通过操作做和更新接口在屏幕上重新显示。

通常来讲,用户只需要将对应的交互操作如按键、触屏等的响应函数与对应的显示功能函数对应起来就行了,但是有些情况下,这样的操作使得系统运行时的开销很大,例如在多级列表时,在显示列表 1 时,选择该列表中的项目 2 时会跳转至列表 2,在列表 2 中选择项目 1 时又会跳转至列表 3……这种情况下,如果用普通的 if-else 或者 switch-case 结构,在当前函数没有运行完成时,又跳转到了一个新的函数中,新的函数没有执行完,又会跳转到另一个新的函数中……而这期间又难免会有一些和外部逻辑器件、传感器等的交互操作。

对计算机系统原理稍有了解的人应该了解,函数调用的过程中、函数参数的传递和函数内局部变量的声明都是要占用栈空间的,每进行一次函数调用、函数的参数、局部变量等就会被压入栈中,函数调用结束后再从栈中弹出。如果出现之前描述的情况,用户编写的函数较为复杂,调用层次又比较多的时候,对栈空间的开销时非常大的。甚至不排除会有内存耗尽的情况。

鉴于此,我设计了这个简单的 HMI 引擎,用来临时的解决一下在这个问题。临时起意的小设计,还不是很完善,欢迎大家吐槽。

3、 原理阐述

为了解决上述问题, HMI 的设计机理就是"无嵌套"概念, 即代码层面的逻辑上, 屏幕画面没有主从之分, 只会从一个画面迁移到另一个画面或者从当前画面返回到之前的画面。所以, 再这个 HMI 引擎中, 用户需要将在自己定义的画面设计、封装好, 然后由HMI 引擎调用。同一时刻被激活并显示的画面中只有一个, 在画面进行跳转之前, 当前显示的画面的数据一定会被释放。

4、 机能简介

HMI 机能的代码实现位于以下两个文件中。

HMI\inc\HMI_Engine.h	数据结构与控制宏定义。
HMI\src\HMI_Engine.c	功能实现。

HMI 引擎的操作函数如下:

HMI_PrepareEngine	初始化 HMI 引擎对象。
HMI_AddScreen	添加用户画面。

HMI_ActiveEngine	激活引擎,同时指定初始画面。
HMI_StartEngine	启动引擎,并使用参数数据初始化并显示初始画面。
HMI_ProcessEvent	执行动作,执行用户、中断或传感器动作。
HMI_PostProcess	后处理,HMI_ProcessEvent 执行成功后需要做的处理。
HMI_Goto	画面跳转。
HMI_GoBack	返回前画面。

HMI 引擎相关的宏定义参数如下:

HMI_SCREEN_HISTORY_MAX	返回历史的保存数量。指用户可以 GoBack 多少次。
HMI_SCREEN_LIST_MAX	HMI 引擎可以最大拥有的画面数。由于同一个画面
	可能在一个操作做流程中进入多次,所以和
	HMI_SCREEN_HISTORY_MAX 参数区别设计。
HMI_SCREEN_START	默认的起始画面索引。

用户再在组织画面时候,需要参照以下结构体的函数指针类型定义,针对每一个画面建立五个函数。

Initialize	画面数据初始化函数
Prepare	画面数据准备/预设定函数
Repaint	画面更新/刷新函数
ProcessEvent	事件处理函数
PostProcess	事件后处理函数

结构体定义参见 HMI_Engine.h 文件中 HMI_SCREEN_ACTION 类型的定义。

```
typedef struct
{

HMI_ENGINE_RESULT (*Initialize)(void);

HMI_ENGINE_RESULT (*Prepare)(const void* pstParameters);

HMI_ENGINE_RESULT (*Repaint)(const void* pstParameters);

HMI_ENGINE_RESULT (*ProcessEvent)(HMI_EVENT_TYPE eEventType, const HMI_EVENT* pstEvent);

HMI_ENGINE_RESULT (*PostProcess)(SGUI_INT iActionResult);

}HMI_SCREEN_ACTION;
```

HMI 中各功能函数的返回值类型均为 HMI_ENGINE_RESULT, 此类型为一个枚举类型, 具体定义如下。

HMI_RET_ERROR	一般逻辑错误。
HMI_RET_INVALID_DATA	无效数据,一般般用于事件处理中。
HMI_RET_NO_SPACE	无可用空间,一般用于 HMI 引擎有初始化处理添加
	画面时。
HMI_RET_MEMORY_ERR	内存错误,如 malloc 失败时。
HMI_RET_UNDEFINED_ACTION	未定义的动作。
HMI_RET_ERROR_STATE	状态错误。
HMI_RET_NORMAL	处理正常结束。
HMI_RET_INITIALIZE	初始化完成。
HMI_RET_CONFIRM	提交处理, 通常指用户触发确认操作, 需要画面迁移
	时。
HMI_RET_CANCEL	取消处理,通常指用户触发取消操作,需要画面回滚
	时。

HMI_RET_FOLLOWUP	需要画面迁移,通常指画面自动发生改变。
HMI_RET_FALLBACK	回滚道初始画面
HMI_RET_INPROCESSING	动作处理中,不能执行新的操作做。
HMI_RET_NOACTION	处理完成,但不需要有任何后续动作。

此定义用户可以根据自己的实际需要重新定义,但需要保留 HMI_RET_NORMAL、HMI_RET_NOACTION、HMI_RET_ERROR 三个关键值,HMI_RET_NORMAL 定义为 0,处理正确时返回值大于 0,错误时返回值小于 0.

最后是 HMI 对象数据结构 HMI_ENGINE_OBJECT 的定义:

```
typedef struct
{

HMI_SCREEN_OBJECT* Screen[HMI_SCREEN_LIST_MAX];

SGUI_INT ScreenCount;

HMI_SCREEN_OBJECT* CurrentScreenObject;

HMI_HISTORY_STACK History;

HMI_ENGINE_STATE State;

}HMI_ENGINE_OBJECT;
```

描述如下:

Screen	指针数组,指向已加入 HMI 引擎的各个画面数据。
ScreenCount	画面总数。
CurrentScreenObject	当前活动画面的指针。
History	画面迁移历史。
State	当前 HMI 对象状态,空闲中或动作处理中。

明确以上定义后,接下来以演示例程中的 HMI 引擎实例和演示例程的 List 画面为例,对照讲解 HMI 引擎的使用。

5、 使用方法

由于 HMI 引擎始终处于动态, 所以演示例程和具体的硬件平台有所关联, 本讲解以模拟器环境为例。

首先,需要定位的时整体的环境入口,在 DemoProc.c 文件中。

模拟器启动并初始化完成时,会调用 InitializeEngine 函数,InitializeEngine 函数中的调用 InitializeEngine_Internal 函数完成 HMI 引擎实例的初始化和添加屏幕动作,整体逻辑处理比较简单,请自行参照源代码。此处不作详细描述。

初始化完成后,HMI 引擎进入等待状态,接下来的所有操作全部围绕 HMI 引擎的 HMI_ProcessEvent 函数, 该函数会调用当前被激活画面对象的 ProcessEvent 函数指针指向的函数,并作后续处理。

打开 DemoProc\src\List.c 文件,此文件为演示程序中列表控件的演示处理。在此文件中,可以中找到前文所述的画面的五个动作的处理函数的定义:

HMI_ENGINE_RESULT	HMI_DemoList_Initialize(void);
HMI_ENGINE_RESULT	HMI_DemoList_Prepare(const void* pstParameters);
HMI_ENGINE_RESULT	HMI_DemoList_RefreshScreen(const void* pstParameters);
HMI_ENGINE_RESULT	HMI_DemoList_ProcessEvent(HMI_EVENT_TYPE eEvent, const HMI_EVENT* pstEvent);
HMI_ENGINE_RESULT	HMI_DemoList_PostProcess(SGUI_INT iActionResult);

以及画面操作函数结构体数据的的定义:

HMI_SCREEN_ACTION s_stDemoListActions =

```
{
    HMI_DemoList_Initialize,
    HMI_DemoList_Prepare,
    HMI_DemoList_RefreshScreen,
    HMI_DemoList_ProcessEvent,
    HMI_DemoList_PostProcess
};
```

和 List 演示画面的定义:

```
HMI_SCREEN_OBJECT g_stHMIDemo_List =
{
    HMI_SCREEN_ID_DEMO_LIST,
    &s_stDemoListActions
};
```

此时,如果用户按下键盘按键,则模拟器会捕捉键盘消息,并组织键盘数据然后调用 EventProcess 函数,由于当前激活画面是 List 画面,所以 HMI 的 CurrentScreenObject 指针指向的是 g_stHMIDemo_List,EventProcess 函数会通过调用 HMI_ProcessEvent 函数,调用 g_stHMIDemo_List 的 ProcessEvent 函数,也就是 HMI_DemoList_ProcessEvent 函数。

HMI_DemoList_ProcessEvent 函数的主要处理如下:

```
switch(*(parrKeyValue+1))
{
    case KEY_VALUE_ENTER:
    {
        eProcessResult = HMI_RET_CONFIRM;
        break;
    }
    case KEY_VALUE_ESC:
    {
        eProcessResult = HMI_RET_CANCEL;
        break;
    }
    case KEY_VALUE_UP:
    {
        SGUI_List_SelectUpItem(&s_stDemoListObject);
        break;
    }
    case KEY_VALUE_DOWN:
    {
        SGUI_List_SelectDownItem(&s_stDemoListObject);
        break;
    }
    case KEY_VALUE_RIGHT:
    {
        Case KEY_VALUE_RIGHT:
    }
}
```

```
if((*(parrKeyValue+0) & KEY_OPTION_SHIFT) != 0)
                          {
                                        SGUI\_List\_SetListItemValue(\&s\_stDemoListObject, \\ s\_stDemoListObject.ControlVariable.SelectIndex, \\
SGUI_List_GetListItemPtr(&s_stDemoListObject,
                                                                                                                    s_stDemoListObject.ControlVariable.SelectIndex)->Valid.Value,
SGUI\_List\_GetList] temPtr(\&s\_stDemoListObject, s\_stDemoListObject.ControlVariable.SelectIndex) -> Decimal.Value-1); temPtr(\&s\_stDemoListObject, s\_stDemoListObject.ControlVariable.SelectIndex) -> Decimal.Value-1); temPtr(\&s\_stDemoListObject, s\_stDemoListObject, s\_s
                          }
                          else
                          {
                                        SGUI\_List\_SetListItemValue(\&s\_stDemoListObject, \\ s\_stDemoListObject.ControlVariable.SelectIndex, \\
SGUI_List_GetListItemPtr(&s_stDemoListObject,
                                                                                                              s_stDemoListObject.ControlVariable.SelectIndex)->Valid.Value+1,
SGUI_List_GetListItemPtr(&s_stDemoListObject, s_stDemoListObject.ControlVariable.SelectIndex)->Decimal.Value);
                          break;
            }
            case KEY_VALUE_LEFT:
                          if((*(parrKeyValue+0) & KEY_OPTION_SHIFT) != 0)
                          {
                                        SGUI\_List\_SetListItemValue(\&s\_stDemoListObject, \\ s\_stDemoListObject.ControlVariable.SelectIndex, \\
SGUI_List_GetListItemPtr(&s_stDemoListObject,
                                                                                                                    s_stDemoListObject.ControlVariable.SelectIndex)->Valid.Value,
SGUI\_List\_GetListItemPtr(\&s\_stDemoListObject, s\_stDemoListObject.ControlVariable.SelectIndex). > Decimal.Value + 1); \\
                          }
                          else
                          {
                                        SGUI\_List\_SetListItemValue(\&s\_stDemoListObject, \\ s\_stDemoListObject.ControlVariable.SelectIndex, \\
SGUI_List_GetListItemPtr(&s_stDemoListObject,
                                                                                                                s_stDemoListObject.ControlVariable.SelectIndex)->Valid.Value-1,
SGUI\_List\_GetListItemPtr(\&s\_stDemoListObject, s\_stDemoListObject.ControlVariable.SelectIndex) -> Decimal.Value); \\
                          }
                          break;
             #if (_SIMPLE_GUI_ENABLE_DYNAMIC_MEMORY_ > 0)
            case KEY_VALUE_F8:
            {
                          SGUI\_List\_Removeltem(\&s\_stDemoListObject, s\_stDemoListObject.ControlVariable.SelectIndex);
                          SGUI_List_Refresh(&s_stDemoListObject);
                          break;
            }
            case KEY_VALUE_F9:
                                                                 // Insert to head.
            {
                          SGUI_List_InsertItem(&s_stDemoListObject, &s_arrstAppendListItems[0], 0);
                          SGUI_List_Refresh(&s_stDemoListObject);
                          break:
            case KEY_VALUE_F10:
                                                                 // Insert to intermediate.
```

```
{
    SGUI_List_InsertItem(&s_stDemoListObject, &s_arrstAppendListItems[1], 5);
    SGUI_List_Refresh(&s_stDemoListObject);
    break;
}
case KEY_VALUE_F11: // Insert to end.
{
    SGUI_List_InsertItem(&s_stDemoListObject, &s_arrstAppendListItems[2], s_stDemoListObject.Data.Count);
    SGUI_List_Refresh(&s_stDemoListObject);
    break;
}
#endif
default:
{
    break;
}
```

可以看到,针对不同的按键处理,画面大的动作和返回值不尽相同,当用户按下回车时,返回值被设定为 HMI_RET_CONFIRM, 按键为 ESC 时,返回值被设定为 HMI_RET_CANCEL, 而其它时候均保持 HMI_RET_NORMAL。

所以,当用户按下回车,EventProcess 中 HMI_ProcessEvent 的最终返回值为 HMI_RET_CONFIRM, 然后将此返回值传入 HMI_PostProcess 后处理函数中继续处理,同理,此函数最终会调用 List 画面定义的后处理函数 HMI_DemoList_ProcessEvent。在这个函数中,如果发现返回值为 HMI_RET_CONFIRM,则认为用户在执行了确认操作,调用 HMI_Goto 函数跳转至相应画面。如果是 HMI_RET_CANCEL,则认为用户要返回,调用 HMI_GoBack 函数,返回之前的画面。

至此, HMI 引擎完成一次完整的动作响应。

6、 写在最后

HMI 引擎的出现,最早就是为了解决我想到的可能出现的栈空间耗尽的情况,将RAM 尽可能多的割让给其他的处理,因为我认为 GUI 其实就是用于向人返回机器的运行结果和接受人的指令,自己本身的存在其实时一种对资源的浪费,如果不不考虑易用性,串口控制台其实是更好的选择。

当然,这个 HMI 引擎这只是对付当时在自己的一时之需,并没有经过更多的斟酌和打磨,只是想到可能会有其他人用到,所以就整合了进来,如果对这方面没有需求, HMI完全可以从 Simple GUI 中剔除掉。

当然,最初创建 HMI 的目的也有提升代码易读性的初衷,但目前看来好像适得其反,应该还是我的设计存在问题吧。当然,后续 HMI 引擎一定会根据 SimpleGUI 的进化而改进的,我也有信心让他称为一个优秀的交互解决方案。

本文仅仅是对于 HMI 的简要说明, 具体内容还请在阅读时参照源代码。HMI 引擎本身用到了大量的函数指针, 只要明白了各个函数指针的意义, 理解起来应该还是比较容易的。

7、 联系开发者

首先,感谢您对 SimpleGUI 的赏识与支持。

虽然最早仅仅作为一套 GUI 接口库使用, 但我最终希望 Simple GUI 能够为您提供一套完整的单色屏 GUI 及交互设计解决方案, 如果您有新的需求、提议亦或想法, 可以联系 QQ 326684221 或电子邮件 xuyulin91@163.com, 也可以在以下地址留言:

SimpleGUI@开源中国: https://www.oschina.net/p/simplegui

SimpleGUI@码云: https://gitee.com/Polarix/simplegui

本人并不是全职的开源开发者,依然有工作及家庭的琐碎事务要处理,所以对于大家的需求和疑问反馈的可能并不及时,多有怠慢,敬请谅解。

最后,再次感谢您的支持。