计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(4):264−267 [doi: 10.15888/j.cnki.csa.006293]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于游戏引擎的交互行为控制器研究①

郝　运[1,2], 郭向坤[2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 沈阳计算技术研究所, 沈阳 110168)

摘　要: 针对游戏中以交互为核心的控制器, 研究了它在实际原型系统中所处的层次, 以及和其它层的关系. 同时分析和补充了硬件指令的传递路径, 进而提出一种基于跨平台开源游戏引擎 cocos2d-x 的实现方法, 并对该引擎做了一定的补充与改进. 该方法通过从控制器视图捕获硬件传来的指令, 以位为单位进行储存与运算, 并通过通道的方式来缓冲并管理命令. 优势在于易于指令的扩展, 并能在语言层级、运行时层级保持实时和高效, 同时支持并行输入, 此外, 还能够很容易应用到现有的项目中.

关键词: cocos2d-x; 控制器; 游戏框架

引用格式: 郝运,郭向坤.基于游戏引擎的交互行为控制器研究.计算机系统应用,2018,27(4):264–267. http://www.c-s-a.org.cn/1003-3254/6293.html

Study of Controller of Interactive Behavior Based on Game Engine

HAO Yun[1,2], GUO Xiang-Kun[2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)

Abstract: Aiming at the interaction in the actual prototype system, this study analyzes the relationship between the level and the other layers. It analyzes and complements the delivery path of the hardware instruction, and proposes an implementation method based on the improvement and expansion of the open source game engine cocos2d-x, which is stored in bits by capturing the commands from the controller view operation. Its advantage to buffer and manage commands over channels is that it is easy to extend the instructions and can be real-time and efficient at the language level, runtime level, support for parallel input, and is easy to apply to existing projects.

Key words: cocos2d-x; controller; game engine

由于游戏、仿真等系统的复杂性, 在实际的开发过程中需要对系统进行清晰合理的划分层次[1]. 其中, 由于输入硬件的差异性, 可能存在多相与并行的控制命令. 现有的游戏引擎, 例如 Unity, 可以方便的添加与删除多个控制器, 但是往往无法从控制信号的底层来有针对性的进行管理[2]. 我们拓展并改进了目前相对流行的开源游戏引擎 cocos2d-x[3], 实现了以交互为核心的控制器, 明确了实际的原型系统中控制层所处的层次以及和其它层的关系. 提出一种实现方法, 该方法具有硬件无关性, 保证了指令系统的可扩展性且易于维护, 并能在语言层级和运行时层级同时保持实时和高效, 为小型游戏、仿真软件等系统提供了一个合理的控制器方案和框架.

1　系统的主要结构

首先, 需要一个外层容器容纳游戏整个生命周期中包括控制层的所有层次, 且此容器为概念容器, 无需被渲染. 结合通用游戏引擎软件开发的理论及实践, 将

① 收稿时间: 2017-07-19; 采用时间: 2017-08-14; csa 在线出版时间: 2018-03-31

具有这样的特征的容器称为“场景”, 进而把上述容器定义为主场景 (MainScene)[4]. 现今, 多数的游戏引擎有相同或相似的概念. 在此基础上, 以“层”为单位进行规划, 主场景包含了主层, 其余层为主层的子层或级联子层, 构成游戏的框架, 如图 1 所示.

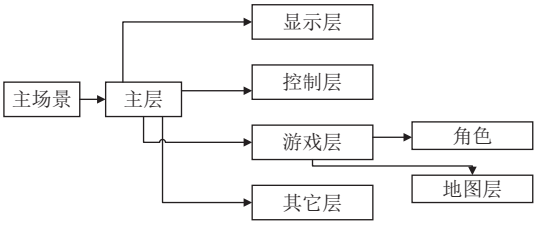

图 1 系统总体结构

定义了系统逻辑上的关系之后, 再围绕游戏的核心框架定义时序关系. 如图 2 所示.

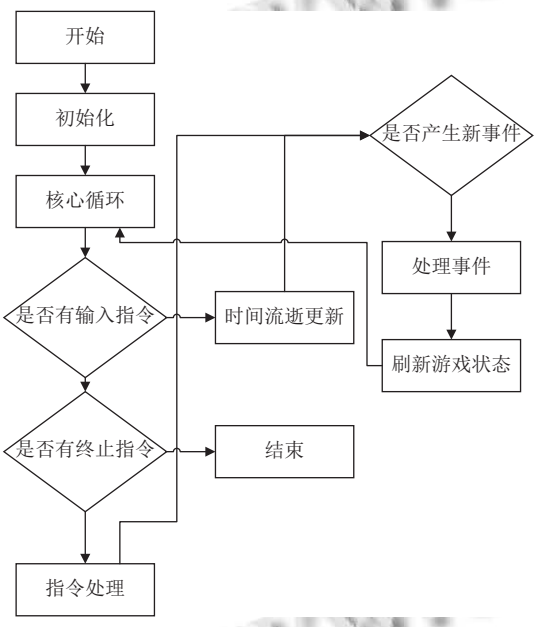

图 2 系统的时序流程

游戏主要部分开始后, 首先进行一系列初始化, 进而转入游戏的核心循环. 在核心循环中, 需要判断是否有激活键值, 即判断用户是否发出了指令, 对该指令进行即时反馈与处理. 同时, 一次次的循环伴随着时间的流逝, 随着时间的推移, 游戏将判断是否触发了新的事件, 对新事件进行相关的处理, 相应地即时刷新游戏状态[5].

2 系统的设计与实现

框架构成的系统采用 MVC 模式, 并明确在 cocos2d-x 3.x 引擎中预定义的概念联系. M(模型) 在系统主体实现中可以认为是各个存储的特定数据, 以及节点的物理实体; 而 V(视图) 可以认为是我们在可视化编辑器中设计的可视部分, 以及各个节点的渲染子层 (通常是 Sprite 类型); C(控制器) 可视为 M 与 V 中的桥梁, 也是控制层中最为关键部分.

在移动平台手机上, 触摸屏幕是操作游戏是基本方式. 所有有效的这一物理操作作用于设备上, 经一系列过程输入给游戏程序, 最后都应该得到相应的输出, 即转化为游戏中对应的实际效果[6].

控制层这个转化过程中的重要的零件. 控制层在视图方面显示了按钮, 此外还作为控制器, 控制器本身并不产生输出或是做具体处理, 而是接受用户的命令并决定是否处理, 交由谁处理.

现以触摸屏幕操作为起点, 游戏中对应的实际效果为终点, 分析操作是如何一步步在内部转化的, 以介绍了控制层实现模式及其工作原理, 定义了控制层的框架和指令数据传递路径, 如图 3 所示.

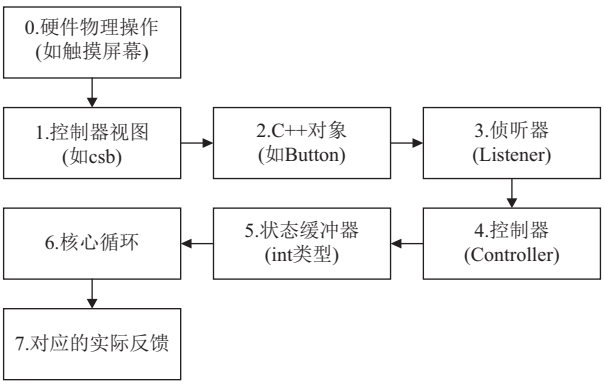

图 3 控制器框架与指令传递路径

将给图中的每一种形式编号, 分别称为阶段 0, 阶段 1, …, 阶段 *N*. 以某个按键按下为例, 阐述阶段 0 是如何推演为阶段 *N* 的, 即在响应状态下某个抽象按键按下这一动作, 是如何转换为游戏中“角色行动”这一实际效果的.

2.1 阶段 0→阶段 4: 控制器前置路径

(1) 阶段 0→阶段 1. 在外部界面设计软件 (如 Cocos studio) 中所设计的按钮即是 V 的典型代表之一. 当左键按下时, 触摸操作这一物理操作由硬件、操作系统等一系列内部过程交由游戏程序处理. 由于控制层位于最顶层, 最后该操作最终作用于在控制层 UI 视

图上.

(2) 阶段 1→阶段 2. 控制层 UI 视图是 Cocos studio 中制作的, 并导出出 csb 二进制文件, 由游戏程序在初始化时导入. 为了使该文件中有关按钮的二进制数据转换为游戏程序可以接受的方式, 使用如下代码:

```
auto btn=static_cast<Button *> (ControlUI→getChildByName("btnName")
```

该语句在代码中定义一个 Button 指针 (即 <Button*>), 并使用 C++中的 static_cast 执行非多态的转换, 令指针指向此按钮, 以后以引用的方式使用此按钮. 系统中存在多个按钮, 可以采用循环的方式进行.

(3) 阶段 2→阶段 3. 在控制层初始化过程中, 为了接受响应, 因此需要注册侦听器:

```
btn→addTouchEventListener(CC_CALLBACK_2(CTRL::BtnClick, this);
```

侦听器的类型有很多种, 对应触摸操作的是触摸侦听器 (TouchListener). 由此, 成功注册侦听器, 按钮被点击后, 侦听器会检测到该事件, 同时调用在注册侦听器时所对应的回调函数.

(4) 阶段 3→阶段 4. 侦听器在回调函数中判断按钮类型, 并通知控制器打开或关闭指令状态集. 有关函数 void ControlLayer::BtnClick(cocos2d::Ref* pSender, TET touchType) 的部分的具体实现如下:

```
if (!m_can_ctrl) return;
if(static_cast<Button*>(pSender)→getName()=="btnLeft")
BTN_SWITCH(CMD_LEFT);
else if
…
```

pSender 是事件的发出者, 这里即指按钮. 按钮的 name 属性标识着各个不同的按钮, 可由 getName() 获得此属性. 究竟是打开还是关闭需要根据按键形式确定, 这样工作由 BIN_SWITCH 实现.

CMD_ON/CMD_OFF 这一对功能模块用于打开或关闭处于 M 层级上的指令状态集. 控制层中控制器的功能可由宏的形式生成分支语句实现, 也可以另外实现行为树或状态机等常见决策模型做出不同的处理.

2.2 阶段 4→阶段 7: 位变量缓冲器

在上一节中, 我们已经实现了控制层的一部分, 即阶段 0→阶段 4. 然而, 指令信息的形式如何以及 CMD_ON/CMD_OFF 的实现机制, 它怎样影响实际游戏, 都是亟待解决的问题.

因此, 我们提出了位变量缓冲器, 其优点在于用单变量就可以存取多个键的键值, 还能改善游戏性能问题, 更有利于识别指令形式, 进而目的性地综合判断处理. 算法的基本思想描述如下, 其中 maxPass 为当前已分配的通道数:

```
procedure BufferBits(stateSet, pass, state)
  while true do
    while Operation is "getPass" do
      if maxPass >= bitsOf(uint) then
        newPass ← 0
      else
        newPass ← 1
        LSHIFT maxPass
      end if
    return newPass
    end while
    while Operation is "readPass" do
      ans ← stateSet AND pass
      return ans
    end while
    while Operation is "setPass" do
      if state = true then
        stateSet ← stateSet OR pass
      else
        stateSet ← stateSet AND pass
      end if
    end while
  end while
end procedure
```

使用“位变量”的方式在一个 int 型变量中存储多个键值. 例如, 取低 16 位分析, 从低位到高位, 每一位分别依次代表一种指令状态. 如 b=0x8888 可以代表指令状态 3, 7, 11, 15 呈现“开”的状态, 其余“关”的状态. 而 c=0x8000 仅有第 15 位为 1, 即“打开”状态, 我们可以用 c 这样的只有对应位为 1 其余位均为 0 的通道去判断 b 中的指令状态 15 是否为打开状态. 当 b&c 位运算结果为非零时, c 通道的指令为打开状态. 此外, 还可以以移位的形式循环处理指令状态.

在 int 类型许可的范围内, 第 i 个按钮的对应通道

可以表示为 (1<<*i*), 各个指令状态在指令状态集中是相对独立的. 使用 CMD_ON/CMD_OFF 宏可以方便的打开一个或多个通道所对应的开关.

“缓冲”是指 CMD_ON/CMD_OFF 并不立即调用执行功能函数, 而只是将到来的命令缓存起来. 当其大体思想如下: 在指令发出时, 并不立刻处理这个指令, 而是将此指令以一定形式存储, 交由游戏的其它模块在合理的时机进行一定的处理. 将“这种形式”即称为指令状态集, 它的作用是识别指令形式和综合判断处理. 这样除了可以改善性能问题之外, 还能连续地处理输出.

游戏核心循环的每一帧的 update 更新函数中按上述算法检索缓冲器, 实现阶段 5→阶段 6 的转换; 当缓冲器打开时, 通知相应的层或结点做相应处理, 以完成阶段 6→阶段 7 的最终飞跃.

由于少量修改了 cocos2d-x 引擎, 因此需要将引擎重新编译, 不能当 cocos2d-x 作为静态链接库.

3 实验验证与分析

为了验证本文设计的框架在不同坏境下的可靠性和效率, 在 CPU 为 ARMv7 架构 2 GHz 主频, 1 GB 内存的安卓手机以及 CPU 为 Intel i3-4170, 4 GB 内存的调试用主机上完成了测试. 测试的目标为一款名为 WAYHOME 的开源游戏, 该游戏使用的原始 cocos2d-x 引擎版本为 3.3, 分别使用典型的自带默认控制器 (称为控制器 A) 和本文的框架下的控制器 (称为控制器 B) 进行测试.

进行安卓平台上的测试. 输入设备是触摸屏. 该设备最大支持 10 点触摸, 但游戏所能接受的输入类型种类为 8. 因此, 进行了 8 点同时触摸测试.

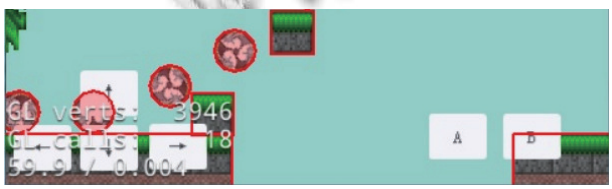

图 4 游戏的 GL 信息

可以看出该场景下, 显卡绘制的顶点数为 3946, 渲染次数为 18, 帧率为 59.9 fps. 控制器 A 和控制器 B 在正常操作的前提条件下, 均不会对游戏性能产生显著影响.

但是在同时发出 8 种指令时, 情况有所区别. 使用控制器 A 的游戏画面出现卡顿, 帧率下降到了平均 20.24 fps, 且游戏中的人物的动作表现不可预见性. 使用控制器 B 的游戏帧率平均为 58.7 fps, 在误差允许范围内, 可以认为没有影响. 此外, 游戏中的人物的表现有符合语义的相对确定性. 例如, “左”和“右”同时按下时, 根据矢量相加的“意图”, 人物在没有外力干涉时应保持静止. 控制器 A 时而左行, 时而右行, 而控制器 B 大多数情况下保持了静止. 可见, 控制器 B 有一定的优势.

4 结语

本文研究并给出了一种基于游戏引擎的控制器框架. 首先分析设计了框架所构成系统的基本结构, 指出了控制层所处的位置以及和其它层方面的逻辑关系. 然后, 围绕系统的核心循环, 分析设计了系统的时序关系. 由硬件底层出发, 在语言和视图的级别追踪和注册了传递控制指令. 通过位变量来储存和管理命令意图, 避免了复杂引用且对程序员在逻辑层面友好, 同时保证了高效率. 通过通道的方式来缓冲输入, 保证在复杂并行输入的同时保持程序的实时稳定性. 最后, 在实际的开源游戏中应用了该框架, 测试结果表明, 相对于原有的默认控制器, 该框架更利于稳定帧率、降低输入延迟以及在极端情况下保证系统的稳定性和表现方面更加符合自然语义.

参考文献

1 Soni B, Hingston P. Bots trained to play like a human are more fun. Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). Hong Kong, China. 2008. 363–369.

2 徐盼欣. Unity 4.x 从入门到精通. 北京: 中国铁道出版社, 2013.

3 周飞龙. Cocos2d-X 引擎中 MVC 框架的设计与实现[硕士学位论文]. 武汉: 华中科技大学, 2013.

4 Wong M, IBM XL 编译器中国开发团队. 深入理解 C++ 11: C++ 11 新特性解析与应用. 北京: 机械工业出版社, 2013.

5 朱成亮. 基于 Android 平台游戏引擎的设计与实现[硕士学位论文]. 淮南: 安徽理工大学, 2011.

6 张萌. 基于动作轨迹识别的游戏控制系统设计[硕士学位论文]. 重庆: 重庆大学, 2013.