存储技术基础 大作业报告

队伍人员

计52 王纪霆 2015011251

计52 于志竟成 2015011275

计52 何欣蔚 2015011253

项目综述

我们基于FUSE，参考了dungeonfs的创意，制作了一个简单的文字解谜游戏框架roomfs。

具体实现的特色功能包括：

- 抽象为文件系统的游戏框架；
- 基于文件操作的交互机制；
- 设计了一套语法，可以使用配置文件灵活实现游戏逻辑，而无需用户修改代码；
- 灵活的架构设计；

以下对系统进行详细介绍。

框架设计

抽象结构设计

我们将解谜类型游戏中的要素归结为以下几个重要结构：地点(room)，物品(item)，响应(reaction)，物品栏(inventory)，全局状态(global state).

地点在游戏中可以抽象为一个有向无环图，mountpoint为其拓扑排序的起点，从该点出发可以通向所有其他地点。这和文件夹的组成形式是完全一致的，因此，我们使用文件夹来代表一个房间。

物品在各种各样的游戏中有着多种用途，既可以查看、使用，也可以获取（加入物品栏），亦或是与其他物品进行组合，甚至可以不是具体的物品，而仅仅表示某个功能（如查看、对话、攻击等）。在这个框架中，物品与文件的概念一一对应。

响应是游戏逻辑的主要组成部分，当玩家对物体进行交互时，便需要触发响应。响应包括先决条件、响应结果两部分，当满足一定条件时，便会触发相应的事件，来推进游戏向下一步进行。例如，当玩家击倒了一个敌人，或是解开一个谜题，通向下一个地点的道路便会打开，等等。在roomfs中，响应由用户编写成脚本，并在对文件的read/write操作结束后触发。

物品栏是构成游戏的另一个主要结构，标志了玩家当前所拥有的资源。一些物品可以被玩家收入物品栏，以供之后使用；一些物品也只有在物品栏中才能够发挥其功能。在roomfs中，物品栏是一个无处不在的文件夹，使得玩家在任何地点都可以访问。

全局状态是用于补全游戏逻辑的机制。全局状态实际上是变量的集合，用于标志需要在游戏中保存的信息，以表明游戏进度。例如，玩家的生命值，玩家的物品收集进度，玩家是否已经触发了某个事件等。这是对用户不可见的，仅有通过响应才能访问和修改。

具体功能设计

由于我们设计的是可以供游戏开发、游玩者使用的框架，以下，我们都严格区分用户、玩家两个概念。**用户**指的是编写脚本形成游戏的使用者，**玩家**指的是探索游戏、完成游戏内容的使用者。

地点

用户可以指定地点名、为地点设置互相连接的拓扑关系，形成玩家需要探索的地图。

每个地点都需要包含一段对地点的描述，玩家可以通过在文件夹下执行`./look`命令来查看。

物品

用户需要指定每个物品的名称、描述，并按照框架设计的语法为其编写响应脚本。

用户可以指定物品的位置，物品可以包含在一个地点下，也可以包含在玩家的物品栏里。可以通过在文件夹下执行`./look item_name`来查看描述。

当物品被玩家执行read/write结束时，均会触发其响应事件。其中，如果玩家使用`./item_name`命令，将可以直接查看物品描述，但shell会先读出文件内容，再调用bash来执行之，故事件会触发两次。

玩家可以使用`echo ";MESSAGE"; >; item_name`向物品中输入一段字符串（例如：输入密码锁的密码，Y/N的选项，对事件的应对选项），输入内容将会被保存、供响应事件使用。

玩家可以使用`./inventory/item1 >; item2`来表示对`item2`使用玩家物品栏中拥有的物品`item1`.这和玩家直接输入字符串相似，内容将会保存并供响应事件使用。

响应

为了满足用户需要的各种功能，roomfs设计了详细的语法，在此不多赘述。用户可以为一个物品编写多个响应，每个响应都可以包含一些先决条件。响应被触发时，其中第一个所有先决条件都满足的响应将被真正执行。

响应可以进行的先决条件判断包括：

- 判断全局状态的值（比较大小、相等）；
- 判断输入是否符合要求（是否为某个特定的物品，是否输入正确）；
- 判断物品栏中是否含有某物品；

响应可以执行的事件包括：

- 为地点添加/删除向其他地点/物品的连接；
- 改变地点/物品的描述；
- 将物品添加到物品栏/从物品栏中移除；
- 清除物品的输入内容；
- 改变全局状态的值（可进行算术运算，完成较复杂的逻辑）

物品栏

在物品栏中的物品将和其他物品的交互方式不同，玩家可在`inventory`文件夹下执行`./check item_name`来查看物品的描述，并执行`./inventory/item >; target_item`来使用物品。

全局状态

全局状态包含int, float, string三种类型，前两种可以进行算术运算，最后一种则不支持。

在物品、地点的描述中均可以直接引用状态，如：

```
You have %d[player_lives] chances left!
```

在显示时，将会使用`printf`来将`player_lives`的值填入描述中。

底层实现

分层设计

我们借用dungeonfs的设计理念，将代码分为三层：由低到高分别是游戏逻辑层、文件抽象层与FUSE层。

游戏逻辑层负责解析用户脚本并实现游戏的基本逻辑，各结构之间的交互机制均在这里实现。

文件抽象层负责将游戏逻辑中的结构包装成文件系统结构，例如为各实体分配inode，实现各实体的read, write, open, getattr等操作。

FUSE层负责接收用户请求，根据其操作和访问对象将请求分配到文件抽象层中的不同实体中。

文件实体

文件系统结构包含了以下实体：

- directory，即地点对应的文件夹；
- inventory，即物品栏对应的文件夹，有且仅有一个；
- file，即物品对应的文件；
- dir_description，即地点下`look`这一用于查看描述的特殊文件；
- inventory_item，即物品栏中物品对应的文件；
- inventory_check，即物品栏中`check`这一用于查看描述的特殊文件

它们各自都分配了不同的inode（根据各自对应的数据结构的指针），分别实现了文件系统的API。这些API将在以下详述。

API使用

为了实现以上功能，我们使用`fuse_lowlevel`接口，使用了以下函数：

- lookup

  用于确认一个文件在文件夹中的属性。地点、物品栏都可以进行此操作，这一操作首先查询文件夹下是否有对应的项，然后调用getattr填充其信息。

- getattr

  用于读入一个实体的权限、类型、大小。每个文件实体都有不同的应对策略。文件夹是相似的，而每个文件都需要填充其文件长度，而由于我们允许文件内容实时根据全局状态计算出来，所以获取长度时也需要先计算出文件内容，才能进行填写。

- readdir

  用于读出文件夹包含的项。对于物品栏，将物品栏中的物品逐个列出(还有check这个例外)填回即可，对表示地点的文件夹，将其连接的其他地点、包含的物品，以及look这个特殊脚本列出填回即可。

- opendir & open

  由于这个文件系统不牵涉到磁盘，所以open没有实际作用，但用于确保打开的文件确实存在，对不存在的文件返回ENOENT.

- read

用于读取文件。这里牵涉到了文件系统的真实实现。实际上实现是很简单的，只是在形式上比较新颖。

读取一个物品文件时，实际上是读出了形如`echo ";item description";`的脚本。这样，直接运行时便显示了其描述。

读取`look`时，实际上是读出了一个shell脚本，用于显示场地的描述。

读取一个在物品栏中的文件时，实际上是读出了预先分配给这个物品的唯一随机数，这样`inventory/item1 >; item2`时，便可以将`item2`中的内容与之比较，判断是否输入了正确物品。

读取`check`时，实际上也是读出了一个shell脚本，用于根据输入参数显示物品栏中各物品的描述。

- release

  对物品文件执行release时，将如前文所述地触发物品上挂载的事件。

- write

  向物品文件执行write时，将会把输入的内容存入文件内置的输入缓冲区，以备进行条件检查。

游戏DEMO

为了显示我们框架的效果，我们自己编写了一些配置脚本，来展示其灵活的功能。

DEMO1：The Room

这是一个简单的密室逃脱类的解谜游戏，包含房间、钥匙、密码锁、机关等等常见的要素。我们录制了一个demo来展现其功能。

DEMO2：Battle

这是另一个例子，用来展示我们的框架不仅可以用在解谜类游戏上，还可以完成一些更加有趣的功能，例如实现传统RPG的“对战”功能。我们同样录制了一个简单的demo，不过考虑到添加更多的功能、更长的游戏流程对于框架而言并没有本质性不同，所以只做了很短的流程以作为展示。

配置运行说明

运行在Linux 4.6.2上，FUSE使用了Github上的最新版本。

在项目文件夹下执行`make run RES_FILE=/path/to/config/file`即可运行FUSE程序，默认挂载到mountdir文件夹下。