TowerFS 文件系统游戏

计 52　于纪平　2015011265

计 55　张瑞喆　2015011371

计 55　王逸松　2015011369

二〇一八 年 六 月

摘要

本课程要求同学们在大作业中开发实现一款和存储技术原理相关的计算机软件。我们选择独立设计和开发一款与传统文件系统所不同的文件系统，这个文件系统参考和借鉴了经典游戏“魔塔”中的游戏机制，结合了打怪升级、获得物品等游戏机制，又综合设计了“传送门”机制，使得整个游戏变得非常有趣。TowerFS 文件系统是我们将魔塔游戏与文件系统相有机结合，开发出的一款基于文件系统的游戏。游戏中的各种操作，均与文件系统中的操作相对应，创意十分新颖。游戏的开发使用了面向对象程序设计技术进行封装，为二次开发和维护提供了便利条件。通过本次作业，我们团队学习到了如何编写一个简易的，而且是自定义的文件系统的技术，又编写出一个有趣的游戏，可以说是一举两得。

关键词：文件系统；游戏；fuse

目录

设计灵感与初衷

在 2017-2018 春季学期，我们三人选修了“存储技术基础”课程，课程要求我们完成一个与存储系统相关的作业设计。在讨论相关方案时，我们一开始希望能实现一个文件系统作为最终的作品。但是主流文件系统都已经有了具体实现，再次实现未免枯燥乏味，没有新意。而进行主流文件系统的开发，往往还面临着开发方面的各种困难，因此，我们认为开发一个新的文件系统并非一个好的选择。

在此基础上，我们考虑以文件系统为基础，做出新的扩展应用，该应用能基于文件系统本身，给使用者提供不同于一般文件系统的体验。我们在讨论后决定，利用文件系统的特性，开发一款不同于通常游戏的一款游戏。

我们选择了魔塔游戏作为原型，我们在讨论中发现了魔塔游戏和文件系统的许多共同之处，并以此为契机，基于文件系统，开发了一款和魔塔形式类似的命令行探险游戏。

《魔塔》游戏简介

《魔塔》游戏是一款经典的策略类的固定数值 RPG 游戏。游戏需要动很多脑筋，任何一个轻率的选择都可能导致游戏的失败。魔塔游戏虽不大，但是制作精美，道具很多，而且难度不低，对智商是一次艰巨的考验。

在《魔塔》的关卡中，玩家需要通过打败怪物、获得道具、打开门

锁、利用道具提高各项参数等方式进行游戏，游戏操作简单，容易上手，又十分有趣。

《魔塔》自推出以来，经过反复的开发，衍生出了同一个游戏框架下的许多游戏版本。0ksh 制作了 RMXP 用的魔塔样板，有兴趣的人可以使用该样板，编辑配置文件，制造出属于自己的魔塔关卡，这一改进使得制作魔塔的能力普及。许多玩家纷纷编写不同的关卡配置文件，使得魔塔的开发得到普及，衍生出许多不同的版本。

设计思路

在经过考虑之后，我们选择设计一个名为《TowerFS》的文件系统游戏，作为课程的大作业。

我们考虑过许多不同的游戏，最终选择了《魔塔》系列，是因为它和文件系统具有许多共通之处，下面一段将结合这些共同关系，叙述整个文件系统的设计思想：

一、 基本设定

文件系统的使用者即对应着魔塔中的冒险者，因此，我们为文件系统的使用者设计了以下基本属性：

- 血量（HP）
- 攻击力（ATK）
- 防御力（DEF）
- 金币（GOLD）

使用者在使用的过程中，与文件系统发生各种交互时，同

魔塔游戏中一样，这些参数会随着使用者进行各种操作，而发生改变。这与游戏中随着玩家操作，游戏角色的各项参数发生改变本质是具有共通之处的。

二、 布局关系

在我们的设计中，一个文件夹的内容和魔塔中一层的内容具有相似性。在魔塔中，到达一层的时候，往往需要玩家先清除某些怪物，然后获得一些道具。再进入到下一层中。而在文件系统游戏中，我们设计了进入文件夹时可以看到一些文件，而在进行了一些操作之后，文件夹中可能会出现一些新的文件，这也与魔塔中的关卡有相似之处。

我们对文件系统游戏进行设计时，简化了这一流程，在进入每个文件夹后，会先出现若干“事件”，通常是文件夹中出现了一些文件，这些文件对应着魔塔中的“怪物”。在打败了这些怪物（对这些文件做出相应的处理）之后，才会出现新的道具或到达新的关卡。

三、 连接关系

魔塔中，通过楼梯可以实现上下楼之间的移动，通过特殊道具可以实现跨越层级的移动。

文件系统中，通过子文件夹可以实现在上下级文件目录中切换，通过链接可以实现跳到其他文件夹的移动。

四、 行动的处理

在文件系统中，通过“cd”命令打开文件夹在我们的设计中

和魔塔中的进入下一层（也有可能是上一层），甚至可能有多个“后继”相类似。通过“cd”本身，也可以使得终端实现到达某一层的操作。

五、 战斗事件的处理

在魔塔中，玩家通过向怪物方向移动，来完成和怪物的对战，具体来说，玩家在怪物面前一格，朝着怪物移动时，会和该怪物发生对战。在对战的过程中，参照回合制对战的方式计算伤害，玩家在每个回合先行攻击。被攻击的一方会损失 $\max(0, \mathrm{ATK} - \mathrm{def})$，其中，ATK 表示攻击方的攻击力，def 表示被攻击方的防御力。直到有一方血量为 0 时结束，血量不为 9 的一方获胜。如果玩家获得胜利，则玩家可以获得该怪物携带的所有金币。如果怪物获得胜利，则玩家死亡，游戏失败。特别地，如果双方都不能对对方造成伤害，则也判定为玩家失败。

而在文件系统游戏中，我们设置了一个“fight”命令来进行这一操作。玩家可以通过“./fight green_slime”命令来和一只“绿色史莱姆”发生对战。对战的记录会被记录下来，并打印在屏幕上反馈给玩家。对战记录中详细记叙了玩家和怪物发生战斗的每一回合发生的事件。

六、 门与钥匙

在魔塔游戏中，门与钥匙是两个很重要的元素，玩家可以通过“钥匙”打开“门”，打开门之后，会进入到一定的区域

内，而这个区域内可能会具有新的怪物和物品。

在我们的文件系统游戏中，也设计有相应的机制。在文件系统中,有些地方可以获取到一定的key,而通过使用这些Key,可以打开相应的一些文件夹（进入到新的空间中)。

七、 权限控制与作弊

为了防止玩家在游戏中，通过强制的修改、删除文件达到作弊的效果。我们的设计中，文件系统的使用者对其中的各种文件和文件夹均无权修改、复制、删除。这也保证了游戏的公平性与趣味性。

执行方法

先按照如下命令安装环境:

```
$ sudo apt install libfuse-dev
$ sudo apt install g++
```

再按照如下命令 build 和执行代码:

```
$ make build
$ make run
```

接下来会生成一个 tower 文件夹，cd 到文件夹中就可以进入游戏了

```
$ cd tower
```

要结束游戏，则需要：

```
$ make stop
```

实现效果

根据以上设计思想，我们队伍对整个文件系统进行了编程。用实现文件系统的方式实现了以下功能，成功制作了一个生动的小游戏。

1、在文件系统中，可以通过“cd”命令通过指定的某个通道。

例如，通过“cd passage1”命令，可以走向当前房间的第一个出口，而出口会通向和这个房间相连接的一个房间。这个寻路系统是由我们系统中建立的双向边表所支持的。

2、在文件系统中，可以通过“ls”命令查看当前房间的怪物。

例如，通过“ls”命令，可以看到当前房间有名叫“Green Slime”的一只怪物。

特别地，如果通过“ll”命令，还能看到隐藏的文件（.status/.fight_log/.open_log/.README）等文件。

3、在文件系统中，可以通过使用“./fight 怪物名称”与怪物

对战。

例如，使用命令“./fight Green\ Slime”，可以和这只绿色的 Slime 对战。对战结果将会输出到终端，同时，保存到“.fight_log”中。

4、在文件系统中，打败怪物后，可以通过“./pickup 物品名称”拾取掉落的物品。

例如，在打败怪物后，使用命令“./pickup YellowKey”，可以捡起黄色的钥匙。或使用命令“./pickup RedStone”，捡起红色的石头（对攻击有加成）。

在拾取钥匙后，通过查看.status 状态，可以看到状态中多了一把黄色的钥匙。如果拾取的是对属性（HP/ATK/DEF）有加成的物品，则会直接对该属性进行加成。

拾取结果将会输出到终端，同时，保存到“.open_log”中。

5、在文件系统中，可以通过“cat .status”查看实时更新的“.status”。

6、在文件系统中，可以通过“cat .fight_log”查看上一次对战的结果。

7、在文件系统中，可以通过“cat .open_log”查看上一次拾取的结果。

8、在文件系统中，可以通过“cat .README”查看帮助文件。

实现原理与代码结构

本次实验的代码使用 OOP 方法编写，代码结构清晰，封装良好，为二次开发提供了很好的平台。

代码全部存储在 towerfs.cpp 文件中，长度约 27kb，另附配置文件（可通过修改该文件，自定义编辑关卡）tower.txt。同时还有用于保存常量的文件 tower-open.txt、tower-fight.txt 等文件。

在实现原理方面，调用了 C++的 fuse.h 功能，重写了文件系统中的 read、write 功能。将这些功能集成到文件系统中。例如，在 cat .status 时，并非真的打开了一个文件，而是由文件系统截获这条消息，然后返回了相应的字符串。这也保证了这个文件是不可修改的。而在打开文件时，由文件系统对路径进行解析，在解析的过程中，如果发现了怪物没有被清除的房间，则禁止路径继续解析，实现了后续关卡的保密功能。

在代码框架方面，我们将所有的内容封装在一个 TowerGame 中，其中包含两个子结构体 TowerConfig（存储初始化得到的游戏配置信息）和 TowerStatus（存储当前游戏状态）。

在 TowerConfig 中，有 4 个 int 类型变量存储了玩家的初始血量、初始攻击、初始防御、初始金币。之后，有 n_monster 变量，表示共有 n_monster 种怪物的类型，之后一个 monsters 向量，存储 monsters 的信息，接着是一个 monster_map 映射表，存储怪物的名字到简称的映射。接着一段描写 obj 的内容，格式与 monsters

类似。之后，有 n_nodes 变量，表示共有 n_nodes 种怪物的类型，之后一个 nodes 向量，存储节点的信息，接着是一个 node_map 映射表，存储怪物的名字到简称的映射。

Monsters 的结构中，包含了名字、简称、HP、ATK、DEF、GOLD，表示怪物的属性。

Node 列表中则存储了名称、事件、连接信息。

事件包括“monster”和“obj”两种可能，还有可能置空“none”。如果是“monster”，接下来会跟着一个怪物列表，表示所有的怪物的名称，如果有多个不同的怪物，系统会自动对它们进行编号。对于“obj”的处理与“monster”类似。特别地，monster 只能出现在 event1 中，而 obj 只能出现在 event2 中。在访问到这个房间时，用户需要依次处理这些事件，才能通过这个房间。

每个连接信息则包括连接类型、连接名称，而每个连接将连接两个房间，文件系统在预处理时，会在这些地方建立双向连接，使得每个房间都可以回到原有的房间。推理可知，这样可以通过来到这个节点的路径回到起点处。

而在第二个类“TowerStatus”中，首先保存了指向其父类的的指针 config。然后，保存了节点信息（因为这些节点可能会发生事件导致参数被修改）。

节点信息 Node 与上面有所不同，下面先保存了 config 配置文件，其中包括两个 event，每个 event 保存了父节点的指针和节点上的怪物、物品信息。

在 config 文件之后，还保存了 player 信息，其中包括玩家的实时血量、攻防、金币、物品等信息。

在这个类中，还存有“readme_file”、“fight_file”、“open_file”等常量信息，其中包含调用这些“文件”时所需要执行的命令。例如，“fight_file”中保存的是：

```
(echo -n fight >> "$1")&&(cat .fight_log)
```

这段代码实际上被放在内存中，且具有执行权限，在需要执行时由文件系统调用，从内存中调取。

未来工作

半个学期的时间过得飞快，我们的作业从无到有，从框架搭建到具体的功能实现，取得了较好的效果。但是，设计中的部分内容并没有被完全实现，例如，对于锁和门的机制虽然已经在框架中预留，但尚未开发。在将来，希望能进行相关方向的进一步开发，完善这一功能。

关于锁功能的详细设计为：在进入 passage 时，需要先调用 NodeLinkStatus 中的“can_reach”函数判定是否能进入到该通道，而当前这个判定是没有对这部分进行判定的，在未来，如果继续这一工作，需要对这个地方添加判断逻辑，确定是否能具有相应的钥匙来通过这一通道。同时需要增加钥匙文件和标记获取钥匙的功能。

总结

在本次大作业中，我们队伍的三位同学都对文件系统的构成有了进一步的深入了解。同时在设计中将所学到的文件系统知识与有趣的游戏相结合，开发了一个这样的非常有意义的作品。在这里，要感谢老师和助教对我们的指导和帮助。课程临近尾声，希望在未来，新的同学能选修存储技术基础课程，继续我们这项有意义的作业。