课程设计报告

181220076 周韧哲

运行方式

本项目选题为植物大战僵尸，基于linux系统开发，请确保安装make, g++ , curses。命令行输入`make run`即可编译运行，或者直接运行可执行文件`./build/Proj-out`。本项目已在ubuntu和manjaro系统经过测试，可以正常运行。如运行遇到问题，请通过QQ联系。

游戏按键操作:

- 按b进入商店购买模式
- 按商店中植物对应的数字键可选择要购买的植物
- 按回车键确认购买
- 按x取消购买，离开商店购买模式
- 方向键可选择格子来种植购买的植物
- 按q可退出游戏

为了能够正确显示图像，请确保**终端的长和宽足够大**，可使用`echo $LINES,$COLUMNS`命令查看，需要确保LINES大于等于40，COLUMNS大于等于113。

目录树

本项目目录树如下:

```
.
├── include/                    #头文件
│   ├── bullet.h
│   ├── common.h
│   ├── courtyard.h
│   ├── game.h
│   ├── input.h
│   ├── plant.h
│   ├── store.h
│   └── zombie.h
├── Makefile
└── src/                        #源文件
    ├── bullet/                 #子弹
    │   └── bullet.cpp
    ├── courtyard/              #庭院
    │   ├── courtyard.cpp
    │   └── grid.cpp
    ├── game.cpp                #控制中枢
    ├── main.cpp                #程序入口
    ├── plant/                  #植物
    │   └── plant.cpp
    ├── store/                  #商店
    │   ├── plantplate.cpp
    │   └── store.cpp
    └── zombie/                 #僵尸
        └── zombie.cpp
```

具体设计

本次项目目标是实现一个基于控制台的植物大战僵尸，支持基本的植物购买与种植以及僵尸的进攻。在本次实验中，我的工作主要包括：

- 实现了植物向日葵、豌豆射手与樱桃炸弹，向日葵每隔一定时间会产生阳光，豌豆射手每隔一定时间会射出子弹，樱桃炸弹种植后会在约4秒后爆炸并消灭上下左右格子中的僵尸。
- 实现了普通僵尸。
- 实现了普通子弹。
- 实现了随机产生阳光并记录游戏分数。
- 实现了每隔一定时间僵尸的一大波攻击。
- 能够在商店购买植物并有冷却时间。
- 能够在庭院种植植物。
- 使用curses来在终端绘图。

我使用一个共同的类LivingObject来作为植物、僵尸、子弹等的基类，因为他们都可以抽象出一些共同的属性：有生命值、攻击值、速度等等，都有坐标，都需要和庭院产生交互，都需要渲染颜色。因而，我使用了枚举类型确定了每一种植物、僵尸以及子弹的类型，并构造结构体InitTable来管理他们的属性的设计数值：

```
enum ObjectType{
    sunflower, peashooter, cherrybomb, zombie, bullet,
};

struct InitTable{
    char name[32];
    int health;             //生命值
    int attack_damage;      //攻击力
    int sun_cost;           //购买所需阳光数
    int prod_sun;
    int cd_time;            //冷却时间
    int speed;
    int act_time;           //隔多久产生动作、速度
    int kill_score;         //被杀可得分数
    int color_pair;         //渲染前景后景色
}static init_table[] = {
    { "SunFlower",  70,    0,  50, 50,  5,  0, 800,  0, YELLOW_BLACK},
    {"PeaShooter",  70,    0, 100,  0,  5,  0,  50,  0, GREEN_BLACK},
    {"CherryBomb",  70, 1000, 200,  0, 10,  0, 200,  0, RED_BLACK},
    {"Zombie"    , 100,   20,   0,  0,  0,  1, 500, 50, MAGENTA_BLACK},
    {"Bullet"    ,   1,    2,   0,  0,  0,  2,  10,  0, GREEN_BLACK},
};
```

在LivingObject中，实现了类内部计数器counter，当counter=0时，当前时刻能够进行动作，如对于向日葵来说是否可以产生阳光，豌豆射手是否可以射子弹，僵尸是否可以攻击或前进（等价于速度）等等。

```
void LivingObject::increase_counter(){
    counter = (counter+1)%act_time;
}
```

由于init_table的存在，可以很容易初始化各种东西，且想修改时较好修改，如SunFlower的构造函数：

```
SunFlower::SunFlower(){
    this->type = sunflower;
    this->health = init_table[sunflower].health;
    this->prod_sun = init_table[sunflower].prod_sun;
    this->act_time = init_table[sunflower].act_time;
    this->counter = 0;
}
```

在商店中，实现了类PlantPlate，用于管理商店中购买植物的相关事项，相当于是商店的商品，并实现了植物的购买冷却效果，Store是它的友元。

```
void PlantPlate::cooling(){
    if(counter > 0){
        inner_counter = (inner_counter+1) % GAME_CLICK;
        if(inner_counter == 0)
            counter--;
    }
}

bool PlantPlate::can_buy(){
    return counter==0;
}
```

在商店中，实现了植物的购买与阳光损耗，注意到，在商店中的购买只会影响对应商品的冷却与总阳光数，而购买植物的真正实例化则是在庭院中的。

```
bool Store::buy(ObjectType type, int &total_sun){
    if(total_sun >= plants[type].sun_cost && plants[type].can_buy()){
        plants[type].buy();
        total_sun -= plants[type].sun_cost;
        return true;
    }
    return false;
}
```

在庭院中，实现了一个类Grid，作为庭院上的每个小格子。它的成员包括了坐标以及在其上的植物或者僵尸等，CourtYard是它的友元。在庭院CourtYard中，成员函数便是Grid，通过宏COURTYARD_ROW和COURTYARD_COLUMN来设置格子数量。

```
class Grid{
    int coord_x, coord_y;
    LivingObject *plant, *zombie;
}
class CourtYard{
    Grid yard[COURTYARD_ROW][COURTYARD_COLUMN];
}
```

类Game是关于游戏总体运行逻辑的类，需要在这里处理其他各种类如Store、CourtYard的交互等。其数据成员有：

```
class Game{
    int score;              //游戏分数
    int total_sun;          // 总阳光数
    int plant_index;        // 当前选择购买的植物类型
```

```
    int cursor_x, cursor_y;  //种植的光标坐标
    bool shopping_mode;      //购买模式
    bool game_lose;
    bool show_cursor;        //是否显示光标
    Store store;
    CourtYard courtyard;
    vector<BulletStruct> all_bullets;
    clock_t last_time, curr_time;
    WINDOW *win;    //curses的窗口
}
```

在main函数中，调用Game的init函数初始化后再调用start就可以开始游戏了。

```
int main(){
    Game g;
    g.init();
    g.start();
    return 0;
}
```

在init中会依次调用store.init()、courtyard.init()以及init_curse()来初始化绘图窗口。游戏进行逻辑是：

- 当curr_time和last_time的差达到阈值后，游戏才开始更新渲染。
- 渲染图像。
- 若玩家输了，打印信息并退出。
- 否则游戏继续，若检测到按键，则处理按键事件，包括移动光标、选择植物、种植植物等等。
- 游戏更新，处理当前状态，并随机产生阳光与僵尸。

```
void Game::start(){
    int tty_set_flag = tty_set();
    curse_render();
    last_time = curr_time = clock();
    while(true){
        wait();
        curse_render();
        if(game_lose){
            print(RED_BLACK, "Lose Game!!!Total Score: %d\n", score);
            refresh();
            sleep(10);
            endwin();
            printf("Lose Game!!!Total Score: %d\n", score);
            exit(0);
        }
        if(kbhit()){
            char ch = getch();
            if(ch == KEYQ)
                break;
            process_key(ch);
        }
        loop();
    }
}
```

其中，游戏更新是本项目的重点，其基本架构如下：

```
void Game::loop(){
    courtyard.check_status(all_bullets, score);
    store.update();
    courtyard.update(all_bullets, game_lose, total_sun);
    gen_sun();
    gen_zombie();
}
```

首先检查状态，检查子弹、僵尸、植物等是否死亡，并更新分数。然后在store中进行更新，其主要是对商品进行冷却：

```
void Store::update(){
    for(int i=0;i<PLANT_NUM;i++){
        plants[i].cooling();
    }
}
```

然后courtyard进行更新，会检测僵尸是否在吃植物，是否有子弹打中僵尸，以及僵尸前进，子弹前进，植物产生阳光，植物产生子弹等等一系列逻辑。

功能亮点

- 使用全局结构体init_table进行初始化数值的存储，直观明了，减少了调试时间。
- 从植物、僵尸与子弹中大量共同的属性抽象出一个基类，减少了代码量。
- 使用继承、友元等OOP方法设计项目。
- 使用curses进行终端绘图，多彩显示。

问题解决

- 如果直接输出到stdout，在终端上的显示由于快速更新而会导致闪屏，因此我使用了curses库来在终端绘图，解决了闪屏问题。
- 每一个类虽然比较简单，但类之间的关系较为复杂，尤其是Game类设计到程序的总体运行框架，因此花费了不少时间设计。

感谢助教的阅读！