

基于《猫武士·新预言》小说原著的 MUD 游戏设计

面向对象程序设计最终报告

nullptr 组

2022 年 6 月 10 日

1 人员信息

组名: nullptr

组员: 略

2 总体介绍

2.1 内容简介

在本次大程序的设计当中,我们选择了 MUD 类文字冒险游戏作为方向,并以《猫武士》(Warrior Cats)小说三部曲——“新预言”(The New Prophecy)为背景,通过面向对象的程序设计方法完成对本次大程序的设计。在游戏中,玩家的最终目标是在地图上保持存活,并收集齐四种族群生活的必需品(Stars)。为了达到这个目的,玩家在游戏中可以通过命令行与程序进行移动、战斗、探索、捕猎、喝水、休息等操作的交互,如此丰富的玩家操作使得我们的游戏具有较强的可玩性。

此外,值得一提的是本游戏的主题框架灵感来源于猫武士官网上一个名为 *The New Prophecy Quest Game* 的 Flash 游戏,目前该游戏已经停止服务。为了使得我们的游戏可玩性更强、设定更贴合原作,在游戏开发中,大部分地形、敌人、角色属性等基础数据均来源于这款 Flash 游戏。

2.2 技术手段与运行环境

本次大程序将采用 C++ 语言编写,严格运用面向对象的设计方法进行程序开发,将游戏中的物品、事件等内容封装为不同的对象,并撰写相应代码来对其中的具体功能进行实现。其中我们使用的 C++ 标准遵循 C++11 的版本规范。

值得一提的是,在本游戏我们使用了 ncurses 字符终端处理库对控制台文字进行输出。相比于一般的字符串输出,ncurses 库支持对文字颜色等属性进行修改,创建菜单、接受方向键输入、可以关闭回显和缓冲等,这便可以使得我们的游戏界面更为美观、具有吸引力。

该游戏是跨平台的,因此我们在设计中使用了 _WIN32 等平台宏来确保平台相关代码不会混用。它可以支持在类 UNIX 与 Windows 两种不同环境下运行,其中类 UNIX 环境需要自行安装 libncurses6, Windows 环境需要自行编译安装 pdcurses。这也是我们选择 ncurses 库进行字符输出的另一个理由: ncurses 库可以在任何遵循 ANSI / POSIX 标准的 UNIX 系统上运行,此外,它还可以从系统数据库中检测终端的属性并进行自动调整,提供一个不受终端约束的端口,这使得它可以在不同的系统平台与不同的终端上表现良好。

为了支持 Windows 终端中文 UTF-8 显示，需要在 pdccurses 编译时加入选项，而且 UTF-8 显示长度与字符串长度不符也带来了麻烦。最终 Windows 的中文支持仍不完善，对话框仍存在显示错位问题，有待改进。在 Windows 下请尽量使用 WSL。

2.3 使用的面向对象和 C++ 特性

- 在游戏中需要放置的各种物品创建了一个抽象基类，并派生出不同的子类，通过虚函数实现多态
- 使用函数模板来实现在地图上放置不同种类的物品（当然用基类指针也是可行的）
- 使用静态类，例如 `MsgBox` 类来实现类似名称空间的功能
- 使用 RTTI 特性获得运行时类型信息，不必显式保存物品的类型
 - 使用 `typeid` 推断类型，例如在放置不同物品的函数模板中允许特别处理 `Enemy` 子类，以便后续操作
 - 使用 `dynamic_cast` 动态向下转换，例如 `Prey` 子类特有的 `hunt` 函数只有在玩家主动捕猎时会调用，否则玩家只会吓走猎物
- 使用 `random` 标准库、`override`, `auto` 关键字等 C++11 特性

3 游戏设计

3.1 类介绍

3.1.1 物品基类: `item`

Class name: `Item`

Private member of `class`:

`name`: 表示物品名称的私有属性（双语）
`counts`, `places`: 表示物品数量、位置等的私有属性
`explore`, `scent`: 表示该物品是否可探索，可闻到的私有属性

Public member of `class`:

`Item(MultiString name, std::vector<int> counts, std::vector<int> places, bool explore, bool scent)`: 构造函数
`getName()`, `&getCounts()`, ...: 获取物品名称、数量等信息的公开方法
`trigger()`: 触发事件纯虚函数

`Item` 类是物品的抽象基类，记录了物品生成和使用的基本属性，以及在猫不小心走上去时会发生的事件 `trigger()`。各种物品实体类直接作为常量硬编码在源代码里，记录初始生成数据。

3.1.2 物品派生类: `benefit`, `defense`, `enemy`, `injury`, `prey`

在五个不同的派生类中，我们用同一个抽象基类 `Item` 对不同地图事件进行了不同派生。它们不仅有自己的构造、析构函数，继承了父类中获取各种信息的公开方法，还对父类中的 `trigger()` 触发器纯虚函数进行了重载。接下来对这些 `trigger` 进行简单介绍：

```
//triggers from benefit.cpp, enemy.cpp, defense.cpp, injury.cpp, prey.cpp
//pseudo code
```

```
void Benefit::trigger(Cat &cat) const
{
    roll effect from dice to deic + 6;
    print benefit info in msgbox;
    if (effect > require) then
        print get benefit info in msgbox;
        switch (applyto)
        case CatVal::health/hunger/thirst: gain health/hunger/thirst and print
            ↪ info in msgbox;
        default: something wrong and print err info in msgbox;
    else then
        reduce attribute and print info in msgbox;
}
```

```
void Enemy::trigger(Cat &cat) const
{
    roll attack and health of enemy;
    first <- true;
    do
        print attack info in msg;
        if (first) then
            create msgbox;
        else then
            int option <- defend or flee;
            if (option == 2) then
                flee and return;
        first <- false;
        roll defense of cat;
        dmg_to_cat <- max{0, attack_of_enemy - defense_of_cat};
        cat lose health;
        roll attack of cat;
        roll defense of enemy from 0 to defense;
        dmg_to_enemy <- max{0, attack_of_cat - defense_of_enemy};
        enemy lose health;
        print lose health info in msgbox;
    while (enemyHealth > 0);
}
```

```
void DefenseAction::trigger(Cat &cat) const
```

```

{
    roll damage in damagerange;
    roll result;
    print defense info in msg;
    result <- max{damage - result, 0};
    print health losing info in msg;
    create msgbox;
    cat lose health;
}

void Injury::trigger(Cat &cat) const
{
    roll damage from 1 to damage;
    print max_damage and injury info in msgbox;
    cat lose health;
}

void Prey::trigger(Cat &cat) const
{
    print scared away info in msgbox;
}

void Benefit::trigger(Cat &cat) const
{
    roll dice to see whether to use the benefit;
    if (dice > require) then
        apply health/hunger/thirst to cat;
        show success in msgbox;
    else then
        show fail in msgbox;
}

```

从上述伪代码中我们可以看出，无论是遇敌战斗的事件、在战斗中防御的事件、遭遇受伤的事件，还是吓走猎物的事件，其触发器都是基于 Item 类的纯虚函数 `trigger` 中实现的。这使得我们的程序实现了动态联编，实现了不同触发器之间接口的统一。这样当玩家在地图上移动时，遇到各种物品，只需要调用 `trigger` 即可实现对应的触发事件。

值得一提的是，对于 Prey 这一派生类，我们的 `trigger` 做的仅仅是弹出敌人已被吓跑的信息。这表明我们倘若直接走到 Prey 所在的格子上会直接把猎物吓跑，而不会进行捕猎。而进行捕猎操作时，调用的是 Prey 类中的另一个方法 `hunt`。

```

//prey.cpp
//pseudo code
void Prey::hunt(Cat &cat) const
{

```

```

    int res <- roll val from 0 to skill;
    print hunt info in msgbox;
    if (res >= skill) then
        print hunt succeeded info in msgbox;
        gain hunger;
    else then
        print hunt failed info in msgbox;
}

```

3.1.3 地图相关类: board, cell, terrain

Class name: Board

Private member of **class**:

screens, totalScreens, preys, injuries;

benefits, enemies, stars, ……: 表示地图上敌人、星星等信息的私有属性

cells[screens][rows][cols], nowScreen, nowRow, nowCol: 表示目前所在位置在地图中
 ↪ 的屏数、行数、列数的私有属性

Public member of **class**:

rows, cols, boxWidth, boxHeight: 地图长度、宽度等基本的公开属性

Board(std::minstd_rand &rng): 构造函数

getCell(int board, int row, int col): 获取 cell 信息

placeItems(std::minstd_rand &rng, std::vector<T> items),

↪ placeTerrain(std::minstd_rand &rng), ……: 对地图进行事件放置等操作的公开方法

clearScreen(), move(int ch): 清屏、移动函数

backupCoordinates(), restoreCoordinates(): 控制当前位置的函数

trigger(Cat &cat): 事件触发器

getScreen() **const**: 获取当前位于第几块地图的公开方法

moveTarget(int ch), battle(Cat &cat, bool fallback), ……: 对地图周围格子进行战
 ↪ 斗、捕猎、探索等操作的公开方法

Board 类保存地图相关的信息，地图由 13 屏组成，每屏 5 行 8 列（原游戏如此）。同时提供了一些方法，以便上层调用来生成地图。

Class name: Cell

Private member of **class**:

visibility: 表示可见性的私有属性

*item, *terrain, *star: 指向事件、地形、星星的私有指针

Public member of **class**:

Cell(): 构造函数

*getItem(), *getTerrain(), getVisibility(): 获取格子上的物品、地形、可见性等信
 ↪ 息的公开方法

setItem(const Item *item), setTerrain(const Terrain *terrain), ……: 设置物品、
 ↪ 地形等属性的公开方法
 renderTerrain(), renderItem(): 渲染格子上的地形、物品的公开方法
 trigger(Cat &cat): 检查格子上战斗状态的触发器
 isEnemy() const: 检查格子上是否有敌人的公开方法
 hunt(Cat &cat): 检查格子上的物品是否为猎物的公开方法

Cell 类表示地图上的一个格子，每个格子都有一个地形，还可能有物品（包括敌人、猎物等）或族群必需品（也就是星星）。

Class name: Terrain

Private member of class:

name, visible, *cells: 表示地形名字（双语）、可见性、所在单元格等信息的私有属性

Public member of class:

Terrain(MultiString name, bool visible): 构造函数
 getName(), getVisible(): 获取地形名称、可见性信息的公开方法
 addCell(Cell *cell), randomCell(std::minstd_rand &rng, Cell *&cell): 随机生成
 ↪ 并设置单元格信息的公开方法

Terrain 类表示地形，还在生成地图时记录了该地形的所有格子，方便后面放置物品。visible 也同理只是为了兼容而保留的，实际都为真。

3.1.4 其他类: cat, star

Class name: Cat

Private member of class:

name, clan, health, hunger, thirst, ……: 表示猫的名字、族群、健康、饥饿等等基础
 ↪ 信息等私有属性

Public member of class:

Cat(), ~Cat(): 构造函数与析构函数
 setClan(char c), setHealth(int h), setHunger(int h), ……: 设置猫的种族、健康等
 ↪ 各种属性的公开方法
 getApprenticeName() const, getWarriorName() const, ……: 获得猫学徒名字、武士名
 ↪ 字等属性的公开方法
 generateStep() const: 生成前进步数的公开方法
 loseVal(), checkDead() const, loseHealth(int h), ……: 扣除各项属性、检查是否死
 ↪ 亡的公开方法
 gainHealth(int h), gainHunger(int h), gainThirst(int h): 增加各项属性数值的公开
 ↪ 方法
 rollVal(CatVal val) const, rollAttack() const, rollDefense() const: 战斗相关生
 ↪ 成随即数值的公开方法

`flee()`, `hasFled()`, `resetFled()`: 与逃跑状态相关的公开方法
`incStar()`, `getDiagonal()` `const`, `getWaterproof()` `const`, `rest()`, ……: 获取星星、
 ↪ 休息等各动作相关的公开方法

Cat 类保存所有角色相关的信息，并且完成角色的各种行为，例如战斗、捕猎、喝水等。在该类中，我们设置了有关上述行为的公有函数成员，方便实现游戏中的相关功能。同时还增加了显示及修改 `stars`, `diagonal`, `waterproof` 等属性的公开方法，方便在类外对这些私有成员进行读取或修改。

3.2 游戏主体部分代码介绍

3.2.1 游戏初始化函数

//game.cpp

```
Game::Game(int argc, char *argv[]) : rng(std::time(NULL)), board(rng),
  ↪ maxStep(5){……}
Game::~Game(){……}
void Game::run(){……}
void Game::parseArguments(int argc, char *argv[]){……}
void Game::readAttributes(){……}
```

上述函数均位于 `game.cpp` 文件中，它们对游戏进行了初始化操作。包括构造函数、析构函数，以及启动游戏的函数，在启动游戏后还进行了参数的解析，向玩家提供展示帮助信息、设置中英文、修改游戏基本设置等功能；随后进行属性的读取，赋予玩家昵称、族群等基本属性。

3.2.2 移动函数

//game.cpp

//pseudo code

```
bool Game::process()
{
  if (can continue to move) then
    if (input == 'q') then
      quit;
    if (input == '\n' || input == '\r') then
      stop, lose val and go next;
    backup current coordinates;
    if (illegal move) then
      go next;
    if (can move diagonally) then
      put terminal into halfdelay mode;
      if (input == 'q') then
        quit;
      switch terminal into normal mode;
```

```

        if (illegal move) then
            rollback and go next;
    if (trigger event) then
        generate step, lose val and go next;
    if (step <- step - 1 and step == 0) then
        lose val;
else then
    do action
    if (quit action) then
        quit;
    generate step;
go next;
}

```

在 `game.cpp` 中, `process()` 函数主要负责处理猫的移动。判断当前状态是否可以继续移动,如果可以则从键盘读入输入,然后执行移动命令(影族斜着走的移动方式通过短时间内同时按下两个方向键完成)。若没有剩余步数,则扣除一定的属性值,玩家可以选择六种行动中的一种(见下文),并生成新一轮的移动步数。

3.2.3 行动函数

//game.cpp
//pseudo code

```

bool Game::doAction()
{
    print the buttons of action;
    choice <- 0;
    for (;;) do
        for (i <- 0 to 2) do
            for (j <- 0 to 3) do
                if (i * 3 + j == choice) then
                    turn on the A_REVERSE attributes;
                    print the selected button in reverse color;
                    turn off the A_REVERSE attributes;
            if (input == 'q') then
                quit;
            if (input == '\n' || input == '\r') then
                break from the loop;
            if (input == KEY_UP/KEY_DOWN/KEY_LEFT/KEY_RIGHT) then
                move up/down/left/right the choice;
    print the blank;
    refresh the screen;
    switch (choice)

```



```

    case 0/1/2/3/4: go battle/check scent/hunt/go explore/rest;
    case 5:
    if (no water) then
        MsgBox print "No water here";
    else then
        go drink;
    go next;
}

```

在 `game.cpp` 中, `doAction()` 函数主要负责对猫的行动进行管理。在控制台的底部,我们会打印出战斗、检查气味、捕猎、探索、休息、喝水六个按钮。玩家可以在六个按钮中选择一个进行行动,而我们的 `doAction()` 函数则会给出响应的控制。同时,和地图移动一样,我们对当前选择的按钮也进行了反色处理,使得玩家可以清楚看出自己目前位于哪个按钮上。而事实上,这些具体的调用都是由其他类完成的,如战斗是由 `Board` 类调用 `Enemy::trigger` 来完成,而检查气味和探索等则完全是地图操作。

3.2.4 渲染状态函数

//game.cpp
//pseudo code

```

void Game::renderStatus()
{
    if (fallback) then
        print the title of all status;
        print the frame of all status;
    switch (clan of cat)
        case thunder/wind/river/shadow: print
            ↪ "ThundrClan"/"WindClan"/"RiverClan"/"ShadowClan";
        default: print "Unknown";
    print the contents of other status;
    print the frame of all status;
}

```

在 `game.cpp` 中, `renderStatus()` 函数主要负责渲染属性栏底下的信息框,即将猫的族群、学徒名、地图编号(原游戏没有)、剩余步数、已收集星数等信息展示出来。

3.3 运行测试

在完成了代码部分的撰写后,我们将游戏正式运行,在运行时,我们的程序会弹出窗口如下:

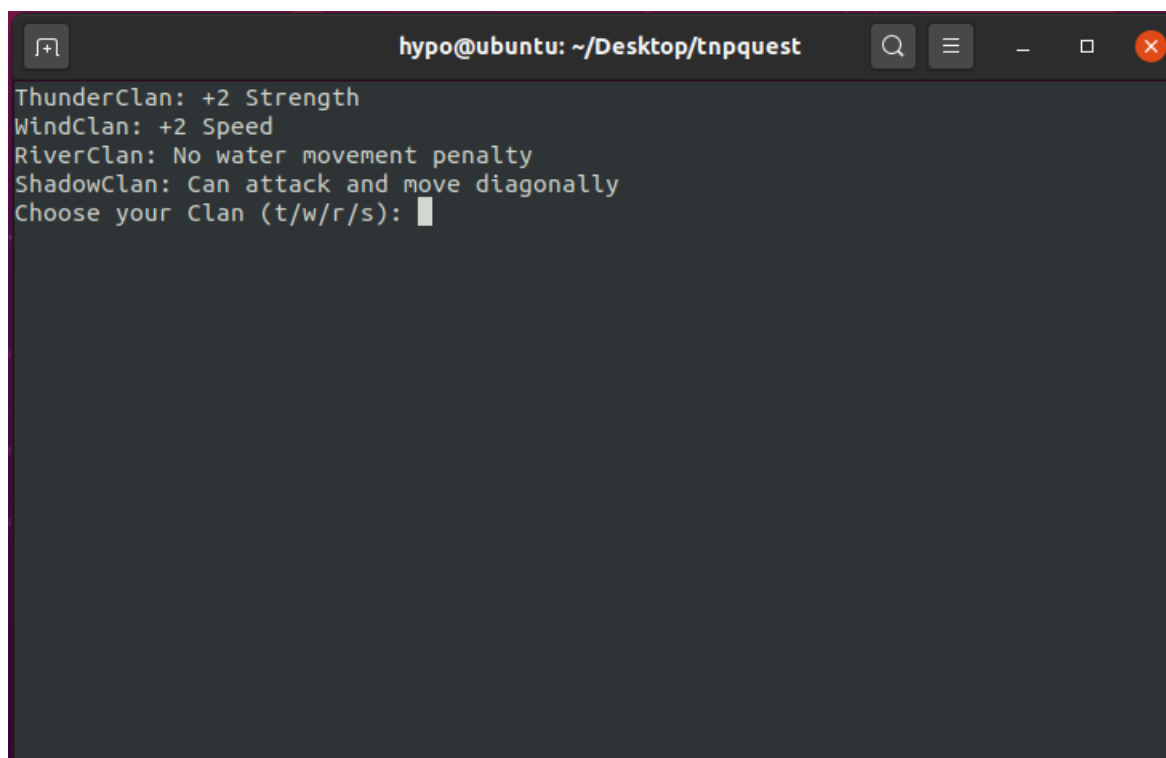


图 1: 启动游戏

随后，我们选择族群并键入名字，游戏会自动随机生成玩家属性和地图，便可开始游戏：

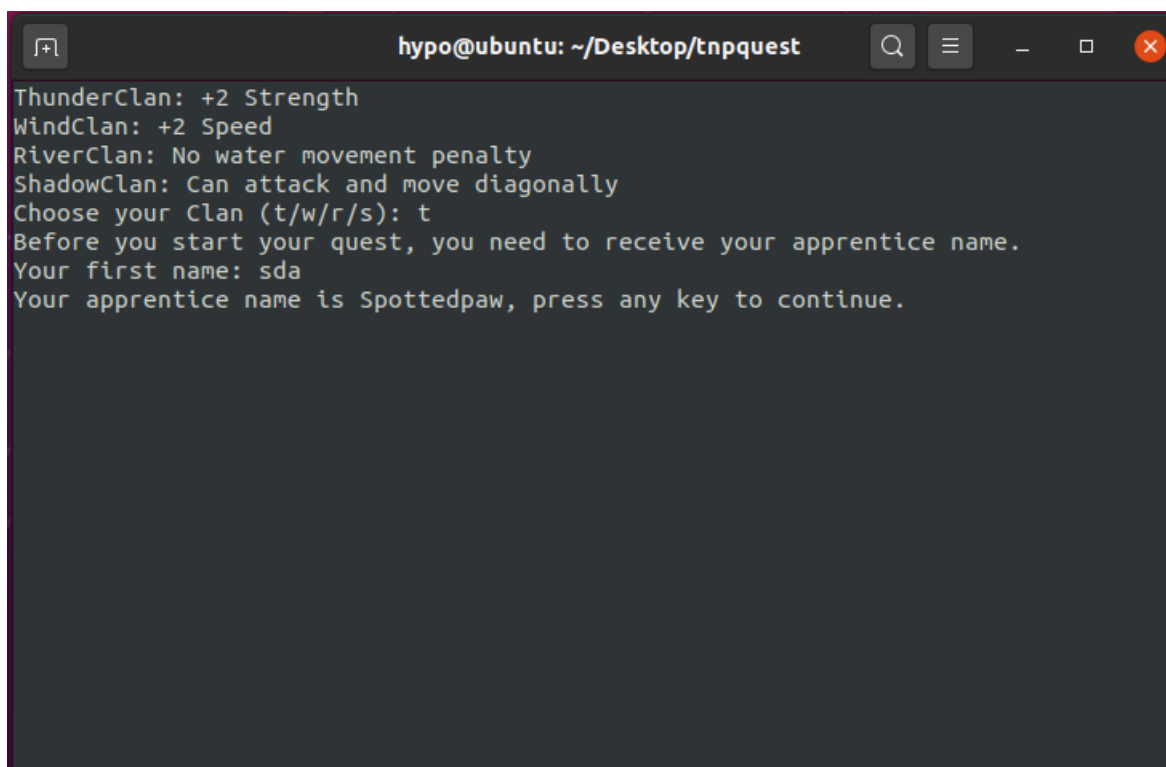


图 2: 输入名字和族群

游戏内战斗相关界面如下，战斗是回合制：

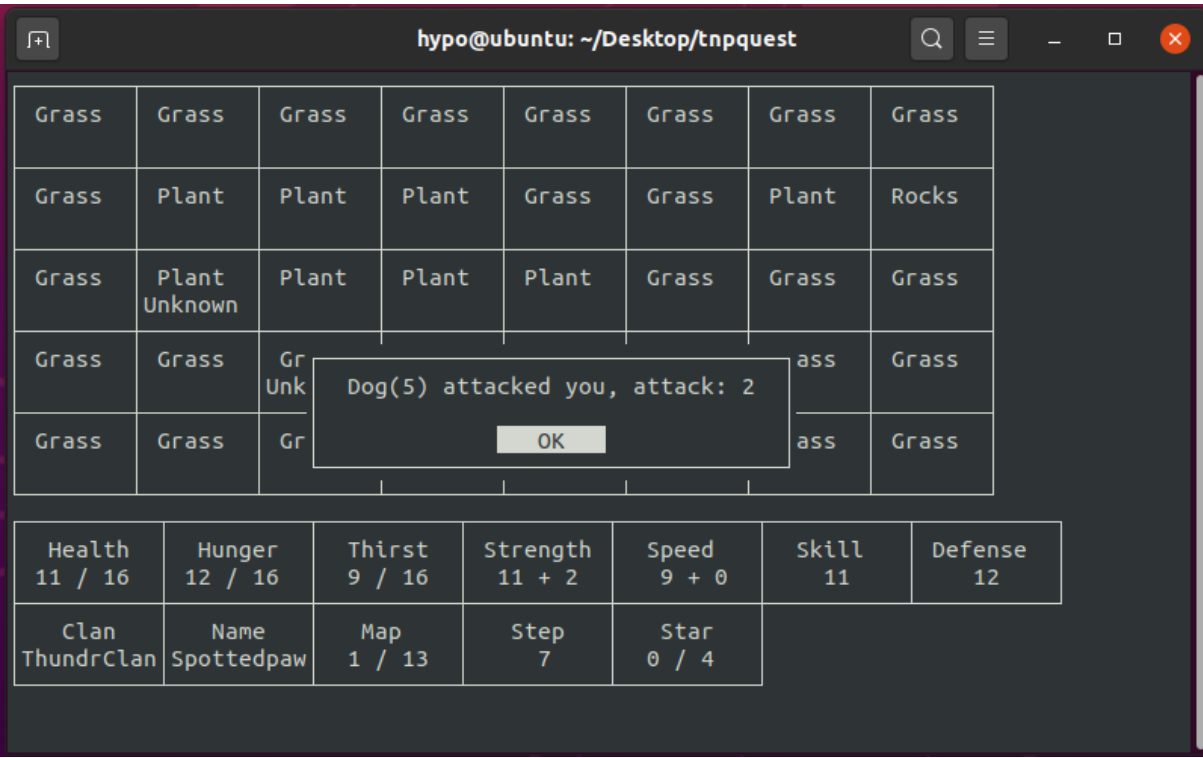


图 3: 战斗界面

在战斗中，如果我们选择逃跑，则会出现如下情况并扣除相应属性：

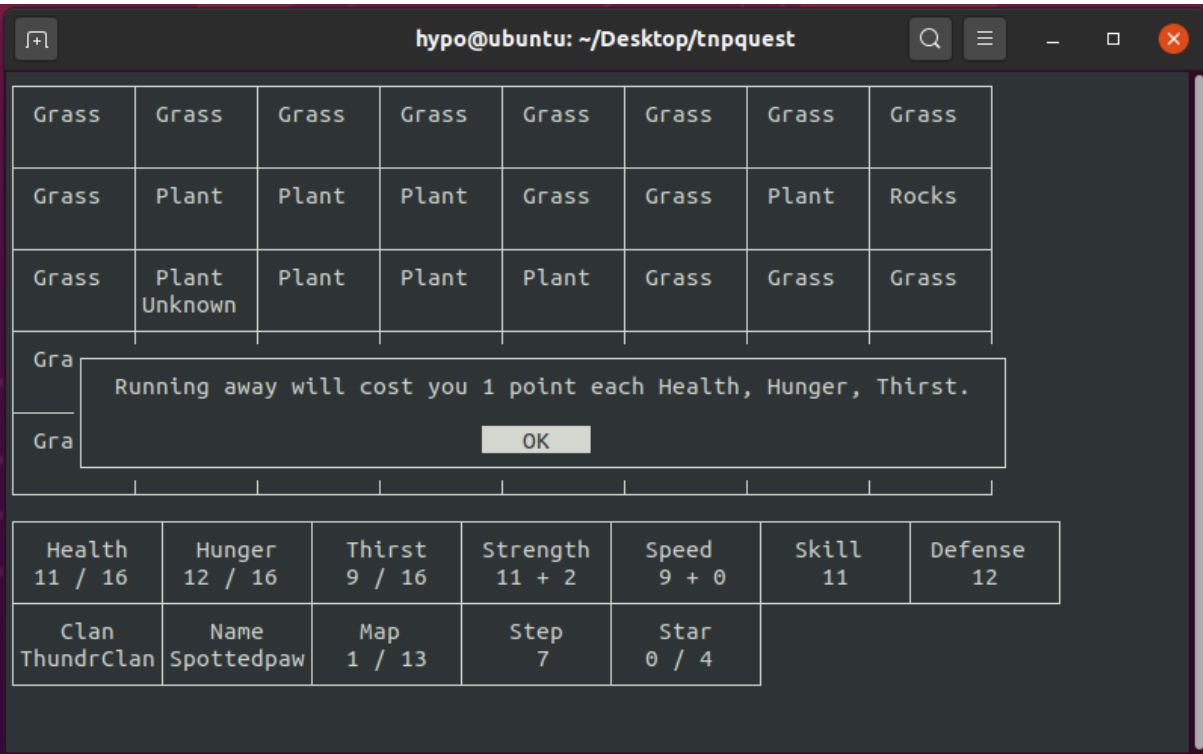


图 4: 逃跑提示

在游戏中，如果我们进行休息操作，则可能获得一定的生命值：

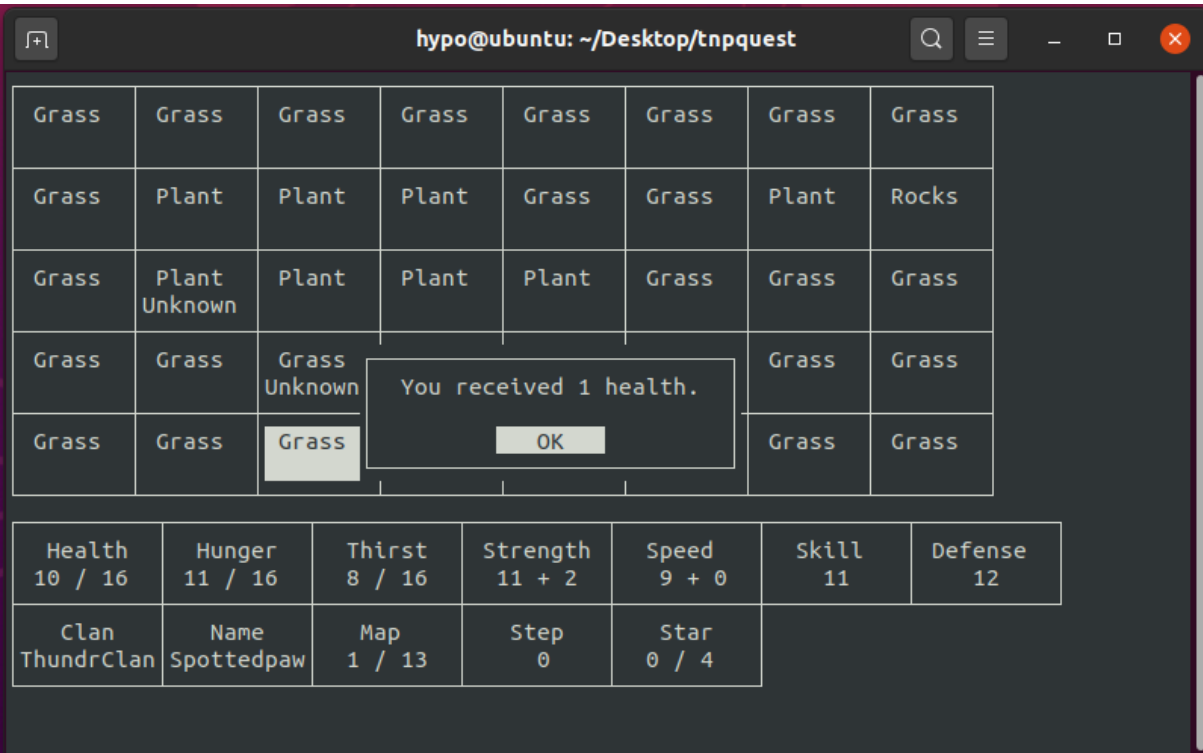


图 5: 休息操作

在游戏中，如果我们进行捕猎操作，则会出现如下情况。如果随机出来的数字比需要的大，则捕猎成功并获得属性，否则不成功：

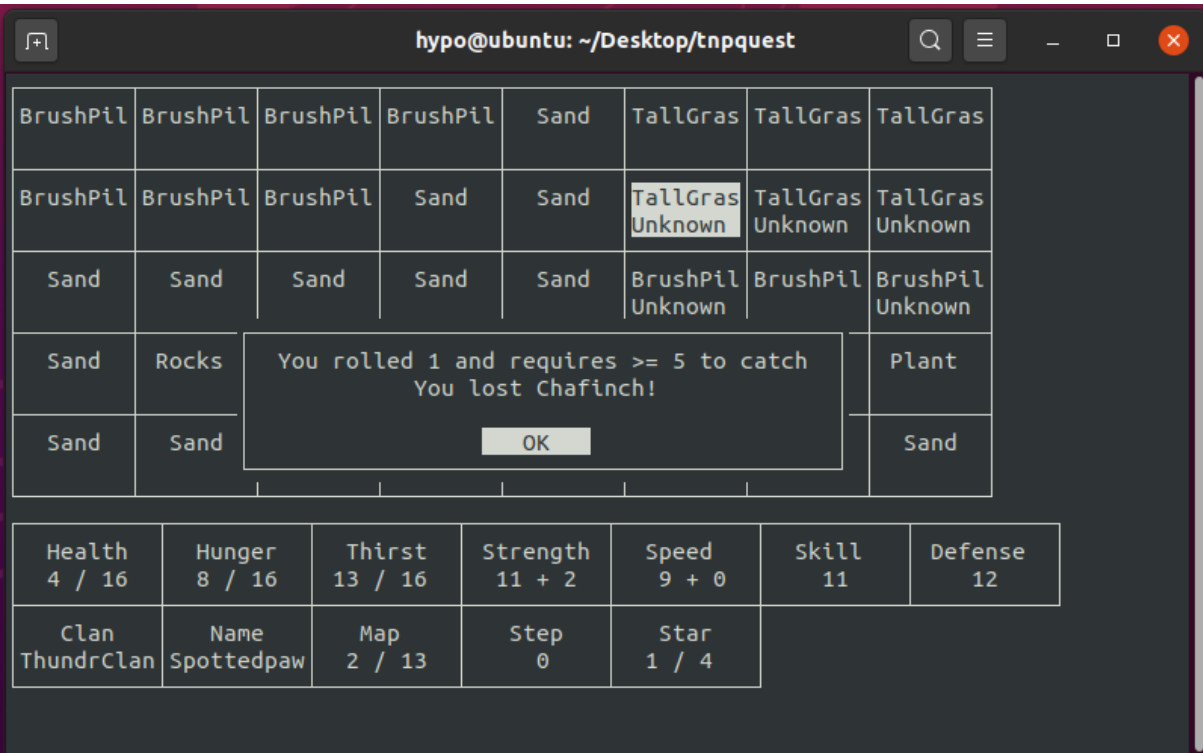


图 6: 捕猎操作

在游戏中，如果我们进行探索地图操作，则可能探索得知一个未知格子的内容：

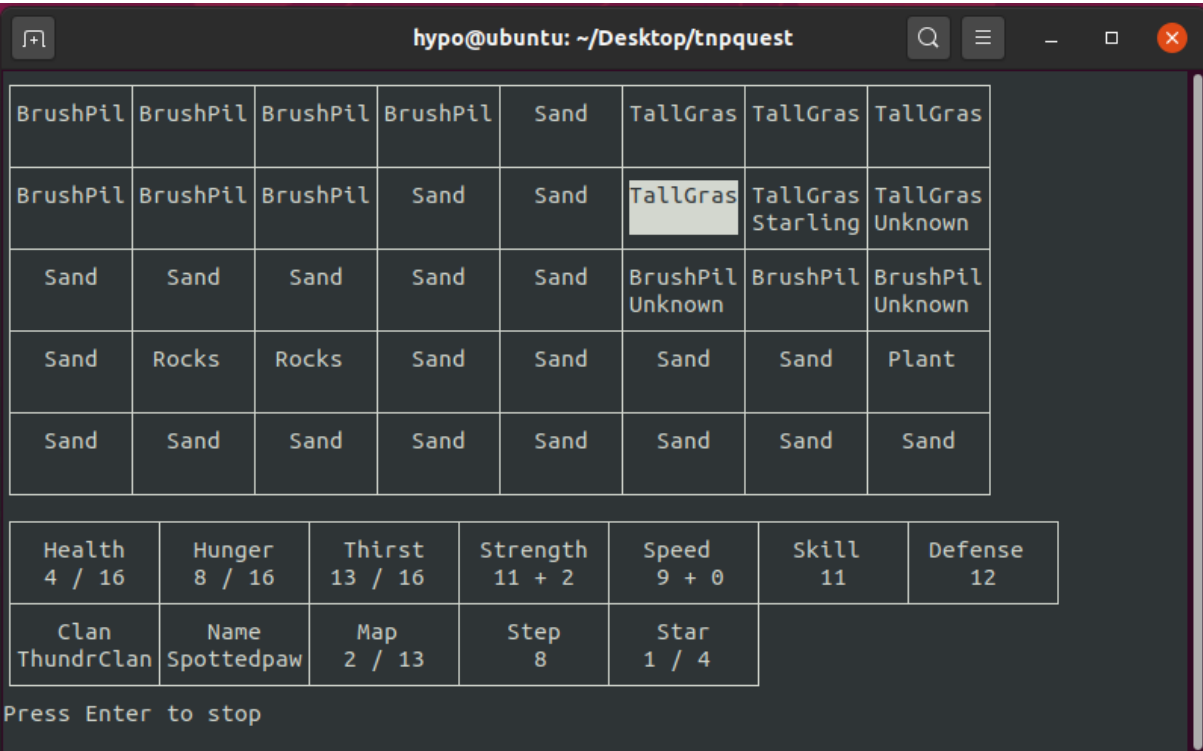


图 7: 探索操作

在游戏中，如果我们在有水的地方进行喝水操作，则会获得渴饮值：

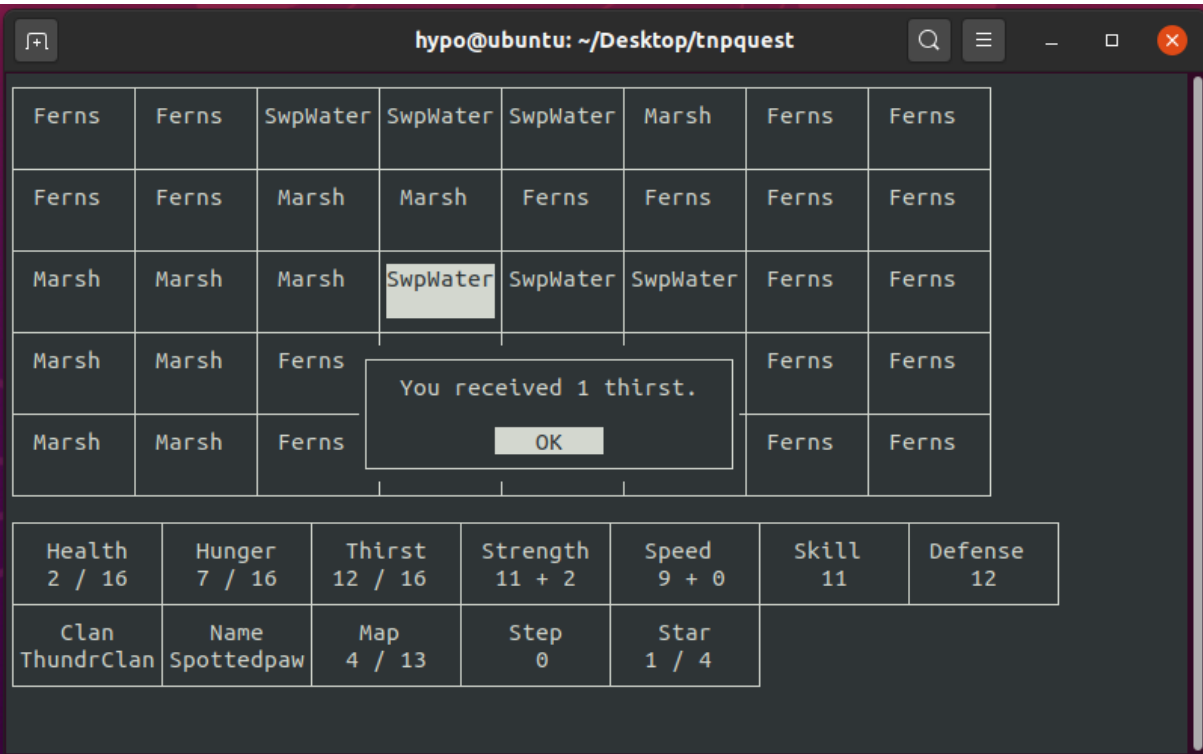


图 8: 喝水操作

在游戏中，如果我们选择检查气味，则会标记出图上所有可以闻到气味的物品：

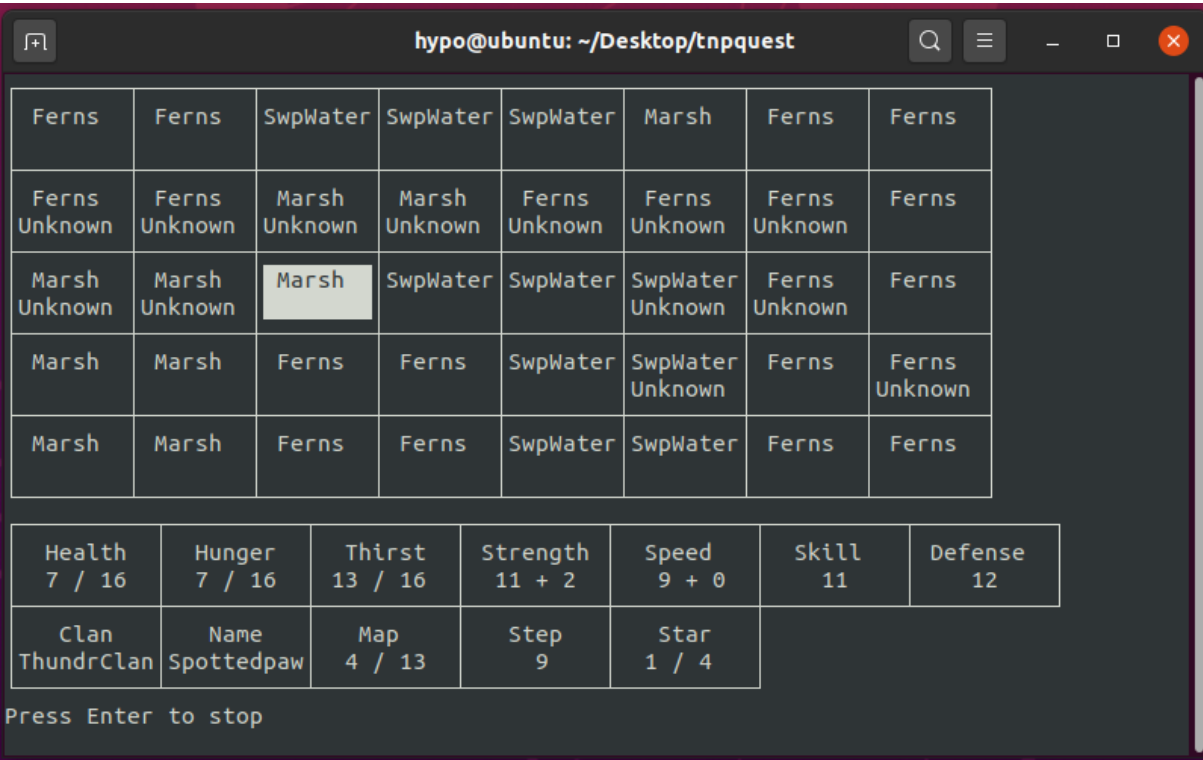


图 9: 检查气味操作

在游戏中，假如某项属性值降为 0 或更低，则会弹出如下信息框，随后游戏结束：

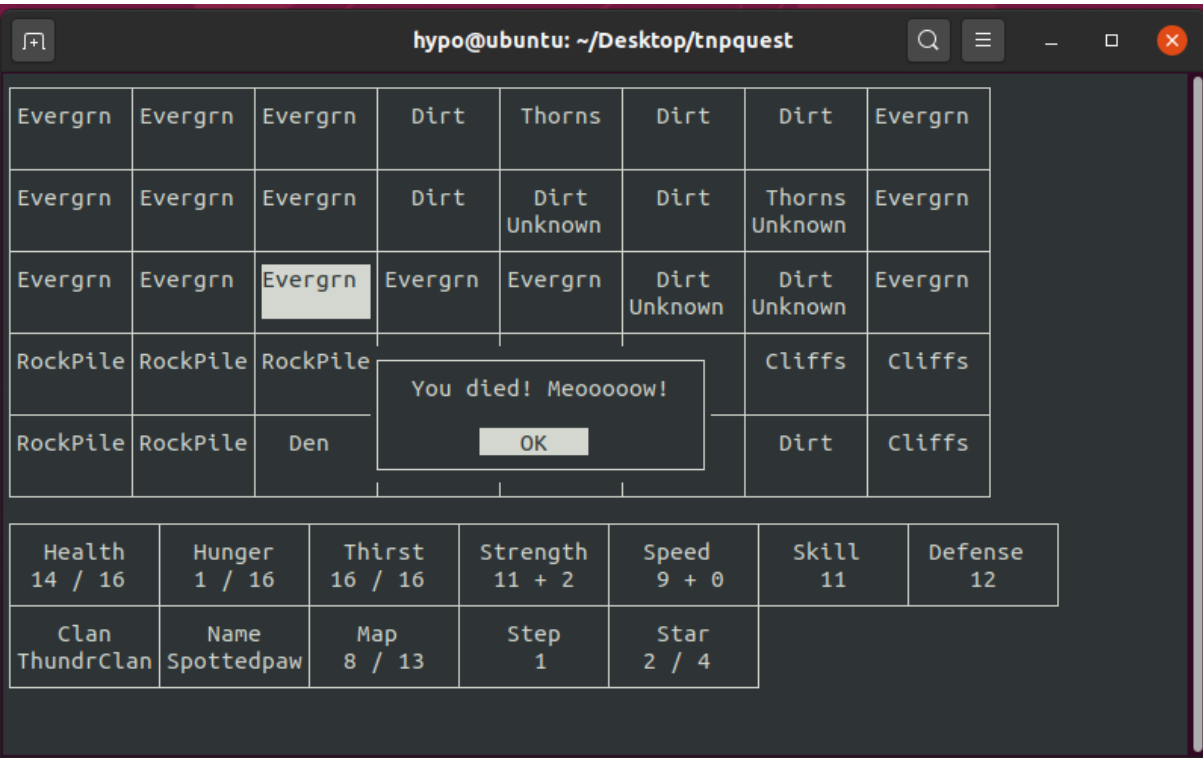


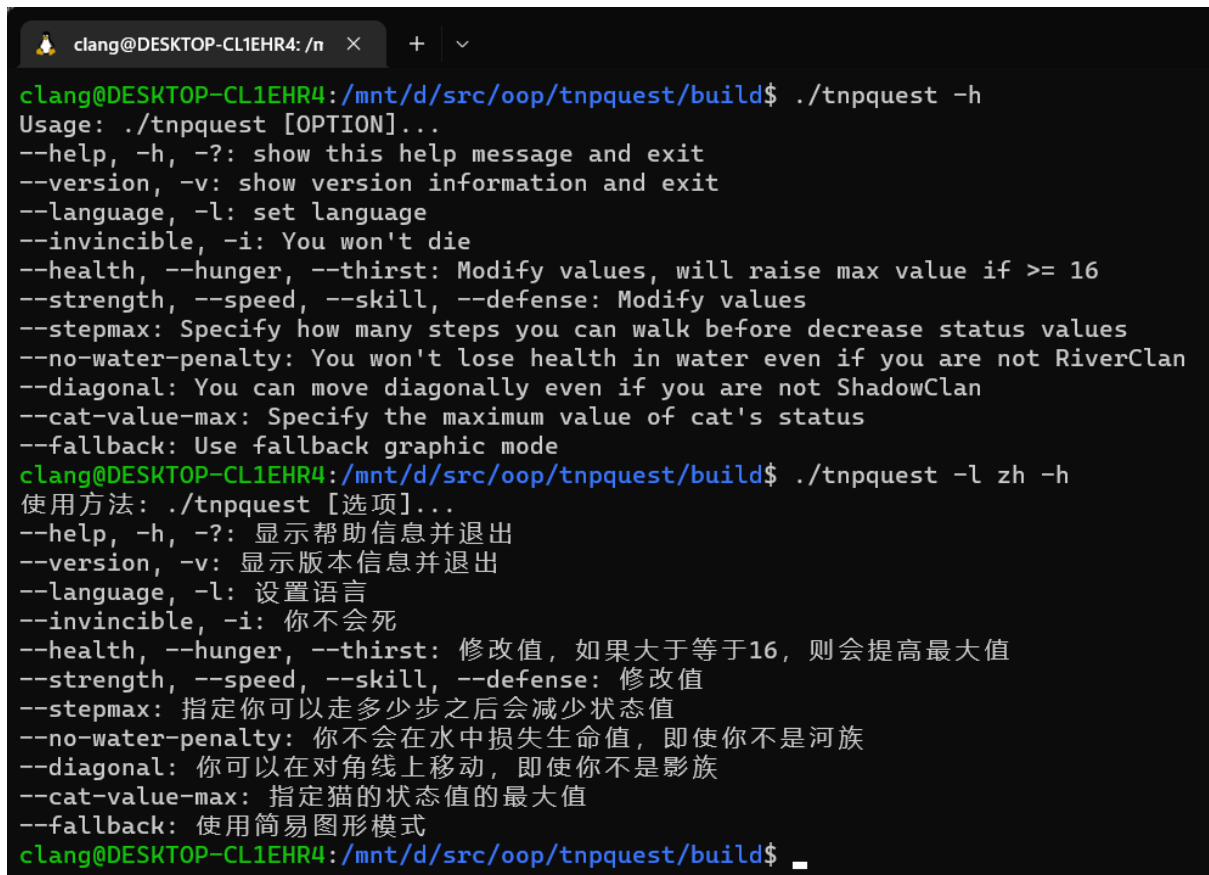
图 10: 游戏结束界面

在 WSL 中，选用中文作为语言，显示出的界面如下所示：



图 11: WSL 中文界面

支持一些命令行选项，可以设置一些作弊选项，可通过 `-h` 来查看帮助，也支持双语：



```
clang@DESKTOP-CL1EHR4: /n  ×  +  ∨
clang@DESKTOP-CL1EHR4:/mnt/d/src/oop/tnpquest/build$ ./tnpquest -h
Usage: ./tnpquest [OPTION]...
--help, -h, -?: show this help message and exit
--version, -v: show version information and exit
--language, -l: set language
--invincible, -i: You won't die
--health, --hunger, --thirst: Modify values, will raise max value if >= 16
--strength, --speed, --skill, --defense: Modify values
--stepmax: Specify how many steps you can walk before decrease status values
--no-water-penalty: You won't lose health in water even if you are not RiverClan
--diagonal: You can move diagonally even if you are not ShadowClan
--cat-value-max: Specify the maximum value of cat's status
--fallback: Use fallback graphic mode
clang@DESKTOP-CL1EHR4:/mnt/d/src/oop/tnpquest/build$ ./tnpquest -l zh -h
使用方法: ./tnpquest [选项]...
--help, -h, -?: 显示帮助信息并退出
--version, -v: 显示版本信息并退出
--language, -l: 设置语言
--invincible, -i: 你不会死
--health, --hunger, --thirst: 修改值, 如果大于等于16, 则会提高最大值
--strength, --speed, --skill, --defense: 修改值
--stepmax: 指定你可以走多少步之后会减少状态值
--no-water-penalty: 你不会在水中损失生命值, 即使你不是河族
--diagonal: 你可以在对角线上移动, 即使你不是影族
--cat-value-max: 指定猫的状态值的最大值
--fallback: 使用简易图形模式
clang@DESKTOP-CL1EHR4:/mnt/d/src/oop/tnpquest/build$
```

图 12: 命令行选项帮助

在地图上移动过程中, 可能会遇到各种随机物品或事件, 包括猎物、伤害、药草、水等, 有的可以探索, 有的不能。下图为成功使用药草增加生命值:



图 13: 随机物品

当找到四个必需品并到达月池时，玩家胜利，并获得武士名：



图 14: 游戏胜利