BIT 2400 - 中级编程

# 教程 4 - 课程

**重要：**

- 不要丢失今天编写的代码。将其保存在某个地方。在接下来的教程中，我们将继续添加这些代码。
- 如果您没有完成实验（实验很多），我将发布每个实验的解决方案，以便您在接下来的教程中使用。
- 你可以从上一个实验室中获取代码开始学习。使用自己的代码会更令人满意，但如果遇到问题，也可以使用提供的代码。使用教程 2 的代码创建 GameGrid 类。

本教程将重点介绍类及其构造函数和析构函数。

## 游戏概述

现在，有趣的部分....，或者说 "有趣 "的部分。在接下来的几周里，我们将逐步制作一款非常基本的僵尸生存游戏。没有人指望一款完美的游戏能让你成为百万富翁，但这项任务具有足够的开放性，初学者只需完成基本任务就能制作一款游戏，而经验丰富的程序员则可以展示他们的创造力和技能。以下是接下来几周的说明：

我们正在制作一款基于文字的僵尸生存 "游戏"，玩家需要携带武器和不同类型的亡灵。这就是我们的目标

## 游戏机制

- 每个玩家都有移动速度和武器类型。你还可以拥有命中率。
- 你有一个起始位置，所有亡灵每回合都会尝试前往你的位置。
- 每回合输入下一个目标位置，然后根据移动速度向该位置移动，或者输入另一个键来使用其中一种武器（"XOR "表示二选一）。
- 您可以让用户拥有所有武器，或者让他们在走过特定位置时拾取武器。
- 每个回合，每个亡灵和玩家的位置都会以网格形式打印出来。输入新位置或使用武器后，下一回合开始。
- 僵尸的超类是亡灵（zombie28 是另一个子类）。吸血鬼和幽灵也扩展（是亡灵的子类）了亡灵。
    - 每种亡灵都有不同的弱点，因此有些武器不会对其造成影响。例如，"木桩 "只会伤害吸血鬼。电锯只影响僵尸....，你会认为它们会 "影响 "吸血鬼，但电影中却没有。
    - 不死的敌人也有不同的速度

- 如果亡灵触碰到玩家（玩家没有正确攻击它），玩家就会死亡。
- 如果玩家使用了影响该亡灵类型的武器，且敌人在武器射程内，则该亡灵死亡。该回合可以有多个敌人死亡。
- 玩家可能知道也可能不知道每个生物是哪种类型的亡灵（但可以通过移动速度推断）。
- 可选项：如果敌人死亡，一些亡灵类型可能会从玩家身边跑开几个回合。僵尸不会逃跑。

- 如果

看起来很简单吧？但到目前为止，还没有人制作出最终的游戏（在之前的学期中，我曾尝试在两个实验中涉及这个内容）。这个游戏的真正意义在于教授编程基础知识，但我还是希望我们能做出一个真正的游戏。

**重要：不要一周又一周地删除您的代码。**我希望你们能不断添加代码，最终制作出这款游戏。确保将代码保存在安全的地方。

**每周我都会发布教程的解决方案。如果你错过了一周或无法完成教程，请随时使用这些解决方案作为下一周的启动代码。**

**我将在早期教程视频中展示游戏的操作视频。**

## A.创建僵尸类

我们即将制作一款基于命令行的游戏，名为 "疯狂僵尸"，第一步是创建敌人：僵尸。

僵尸在场地中行走（有 X 和 Y 位置），速度可慢可快。它也可以是刚死的（还能吃人），也可以是死透的（不能再吃人，但可以移动）。僵尸也会试图攻击玩家，如果它们到达玩家身边，就会发动攻击。

- 创建 Zombie 类。
- 根据上述信息（4 个属性）为该类创建**受保护**属性，再加上 2 个属性：目的地 x 位置和目的地 y 位置。这就是僵尸要去的地方（慢慢地）
- 创建 4 个功能：
    - **setDestination** 获取两个整数参数（xPos 和 yPos），并使

僵尸的理想地点。

- 将僵尸移向目标或目的位置：该方法应称为 **takeTurn**，没有参数。该函数返回一个整数，表示僵尸对玩家造成的伤害（因此大多数情况下你期望的伤害值为 0）。

- 打印僵尸的位置和一些相关信息。printInfo 不返回任何信息，也不接受任何参数，而是使用 cout 语句将信息输出到屏幕上。
- 函数 **hit** 不接受任何参数，而是返回僵尸造成的伤害值（比方说，返回值 3 表示玩家的 "3 点命中率"。如果僵尸已经死亡，则返回值为 0）。请注意，这个值随后会从 takeTurn 返回。
- 您可能还应该为各种属性创建获取方法。您可以选择其他类需要访问的属性。

这是一个游戏

### 移动

- 移动僵尸的速度应受到限制。
- 比方说，典型的僵尸每回合走 3 "步"，每一步在 x 或 y 方向上超过 1 个位置。因此，从（3，3）移动到（8，3）就意味着僵尸在一圈后位于（6，3）。
- 从（4，3）移动到（1，2）可以将僵尸移动到（4，2），但直到下一个回合才会移动到（2，2）。
- 快速僵尸（我们将在下周制作快速僵尸课件）可能可以在一个回合内从 (4, 3) 跑到 (1, 2) 的 6 个位置，但不能从 (1,1) 跑到 (8, 9)。
- 如果僵尸尚未到达所需的位置，**takeTurn()** 函数会更新僵尸的位置。因此，在上面的快速僵尸案例中，takeTurn 会将僵尸从（1,1）移动到（4,4），然后在下一轮移动到（7,7）。再转一圈，它就会停在（8,9）处。

创建一个包含类定义（包括函数原型）的 .h 文件和一个包含函数定义的 .cpp 文件。

确保测试僵尸类是否能正常工作。给它一个目的地并调用几次 takeTurn，在每一轮之间打印僵尸信息。

## C.创建主文件

您可以在您的.cpp 文件中使用以下代码来测试您的代码。不过，您需要在下面的代码中调用任何您尚未实现的函数。这只是为了让您能更快地测试您的代码。

```
#include
"zombie.h"#include
<iostream>      using
namespace std;
```

```
void main()
{
        cout << "1. 僵尸 z1" << endl; 僵尸
        z1; z1.setDestination(3, 4);
        z1.printInfo();

        cout << endl << "2. Zombie z2" << endl;
```

```
    Zombie* z2 = new Zombie(8, 8);
    z2->setDestination(5, 5);

    cout << endl << "3. 僵尸 z3" << endl; 僵尸
    z3(22, 2, shotgun); z3.setFast(true);
    z3.setDestination(2, 2);

    // 再更新一次位置 z1.takeTurn();
    z2->takeTurn();
    z3.takeTurn();

    删除 z2;
}
```

输出结果如下

```
1. 僵尸 z1 构造函数:
zombie()
僵尸现在在 (0,0) 位置，前往 (3,4)

2. 僵尸 z2
构造函数: zombie(int, int)
僵尸现在位于 (8,8) 位置，前往 (5,5)

3. 僵尸 z3 构造函数: 武器 (
int)
构造函数: zombie (int、int、int)
僵尸现在位于位置 (22,2) ，前往位置 (2,2) 。

僵尸现在位于位置 (2,1)。僵尸现在位于位置
(6,7)。僵尸现在位于位置 (16,2): ~zombie()
析构函数: ~zombie() 析
构函数: ~weapon() 析构
函数~zombie()
按任意键继续 . . .
```

观察前面的输出，试着理解哪个调用对应哪个僵尸（z1、z2 或 z3）。

## D.创建武器类

没有武器的僵尸有什么意义？每种武器都有类型（int 或枚举类型）、射程（int）和威力（int）

- 创建武器类。

- 根据上述信息为该类创建属性（3 个属性）。
- 创建相应的功能：
    - 构造函数和析构函数

        - 默认值应该是多少？
        - 创建一个以武器类型为参数的构造函数（并根据参数确定射程和威力）。
    - 每个属性的**获取/设置函数**。与属性相关联的 "getter "函数返回该属性，而属性的 "setter "函数将一个新值作为参数，并用这个新值更新属性。
        - 例如：int getRange() 和 void setRange(int newRange) 可以是范围属性的获取器和设置器。一个用于设置武器的射程，另一个用于获取该值供其他对象查看。
- 为每个构造函数、析构函数和函数写一个**临时**输出语句，并注明函数名称及其参数。例如，默认情况下这将是武器构造函数的输出。
    - cout << "constructor: weapon()" << endl;

完成本实验后，您应该删除这些语句。

创建一个包含类定义的 .h 文件和一个包含函数的 .cpp 文件。

现在，在你的主方法中创建一个武器和一个僵尸，然后分别调用各种函数来测试你的代码。

## 武器类型

我在 Weapon.h 中创建了一个名为武器类型的枚举数据类型。

```
类型化枚举
{
    猎枪 = 1、
    黄油刀 = 2、
    卡塔纳 = 3
} weaponType;
```

这意味着您可以写 shotgun 来表示 1，而不是使用值 1。开关语句可能是这样的

```
开关
 {
    霰弹枪:
      范围 = 3;
      force = 3;
      break;
 (.....):
      范围 = 1;
```

您不必采用这种方法，但这是一种有用的策略。您也可以将 SHOTGUN_ID 定义为 1，将 KATANA_ID 定义为 2，等等。

## E.修改僵尸类

修改僵尸类，加入武器变量。等等，僵尸有武器？僵尸有武器？现在是有的（下周我们将详细讨论这个问题）。这个武器是一个动态对象（指针）。你应该使用 "new "和 "delete "来创建和删除它。只有在必要时才创建它。

- 添加属性
- 创建相应的函数（get/set）
- 创建/更新僵尸的默认构造函数和析构函数，并创建另外两个构造函数，一个以 xy 位置为参数，另一个以 xy 位置和武器类型为参数。
- 每个构造函数和析构函数都有一个输出语句，其中包含函数名称及其参数。例如，默认情况下这将是僵尸构造函数的输出。
    - cout << "constructor: zombie()" << endl;
- 在每个构造函数和析构函数中添加 cout 行，以观察调用顺序。
- 继续根据这些新变化测试您的代码。

## F.迁移游戏控制（如果你愿意，可以推迟到 L07 再做）

选取教程 2 中的代码并进行修改，以创建一个 GameGrid 对象。GameGrid 对象是我们创建的游戏的全部。它将处理用户输入、向用户显示网格以及更新位置。

- 请记住，一旦将这些函数放入一个类中，您就不需要这么多参数了；玩家和敌人的位置将作为成员变量存储。
- 如果您想确保自己了解对象和信息隐藏，这是一项很好的任务。GameGrid 必须了解各种对象（如移动器、玩家和敌人），并与它们通信，以移动敌人、让玩家攻击等。
- 我创建了一个移动器类，每个敌人类和玩家类都将其作为成员对象。你可以告诉你的移动器对象移动到位置（5，6），并给它一个速度，然后该对象会在每一回合适当调整位置，直到玩家或敌人到达位置（5，6）。这个移动器类确实简化了我的代码，并使代码可重复使用。

- 鉴于本实验的长度，我将创建这个类，你可以使用或修改解决方案。这只是我的实现方法。你还可以采用很多其他方法，但如果你看了我的代码，就会知道我是如何访问成员变量、如何传递信息以及如何将复杂的问题分解成较小的独立代码块的。你应该能在以后的作业中使用这些代码。

另外请注意，我 "重构 "了教程 3 的代码，删除了硬编码值，使代码更加模块化。

## 提交作品

确保在 Brightspace 上提交您的作品。请随意使用下一个教程中发布的解决方案（它是毕竟这是一个漫长的实验室），但你一定要尽可能多地完成这些工作。