项目 3

寻宝游戏实现

项目描述

习近平总书记在二十届中共中央政治局第十一次集体学习时强调，发展新质生产力是推动高质量发展的内在要求和重要着力点。习近平总书记指出，新质生产力是创新起主导作用，摆脱传统经济增长方式、生产力发展路径，具有高科技、高效能、高质量特征，符合新发展理念的先进生产力质态。习近平总书记指出，科技创新能够催生新产业、新模式、新动能，是发展新质生产力的核心要素。必须加强科技创新特别是原创性、颠覆性科技创新，加快实现高水平科技自立自强，打好关键核心技术攻坚战，使原创性、颠覆性科技创新成果竞相涌现，培育发展新质生产力的新动能。

我们要积极响应国家对技术型人才培养的号召，通过实践与创新相结合的方式，推进信息技术教育的发展。本项目以寻宝游戏为载体，旨在深化学生对面向对象编程理念的理解和掌握。我们将运用 Python 中面向对象和类的相关知识，设计并实现一个寓教于乐、富有挑战性的寻宝游戏。此项目不仅致力于提升学生的 Python 编程技能，更着重于激发学生的探索欲望和创新精神，培养学生解决问题的能力，以及面对挑战时的创新思维。在寻宝游戏的探索过程中，学生将体会到勇于尝试、不断探索的乐趣，从而更加积极地投身于编程学习和科技创新的实践中。

拓展读一读

中共中央政治局 2024 年 1 月 31 日下午进行第十一次集体学习，内容是加快发展新质生产力，扎实推进高质量发展，目的是结合学习贯彻党的二十大和中央经济工作会议精神，总结新时代高质量发展成就，分析存在的突出矛盾和问题，探讨改进措施，推动高质量发展取得新进展新突破。

会议的主要内容可以概括为以下几点。

（1）高质量发展：会议强调了自党的十八大以来，中国在全面贯彻新发展理念的过程中，经济发展阶段性特征和规律的深化认识，以及高质量发展的重要性。

（2）科技创新：会议提出了科技创新是发展新质生产力的核心要素，特别是原创性、颠覆性科技创新的重要性。

（3）新质生产力：会议介绍了新质生产力的概念，它是由技术革命性突破、生产要素创新性配置、产业深度转型升级而催生，以劳动者、劳动资源、劳动对象及其优化组合的跃升为基本内涵，以全要素生产率大幅提升为核心标志的当代先进生产力。

（4）创新驱动：强调了创新在新质生产力发展中的显著特点，包括技术、业态模式、管理和制度层面的创新。

会议还提到了推动高质量发展面临的外部环境挑战和内在条件限制，以及需要解决的问题。最后，会议强调了发展新质生产力是推动高质量发展的内在要求和重要着力点，并提出

了具体的改进措施。这些措施包括大力推进科技创新、以科技创新推动产业创新、着力推进发展方式创新、扎实推进体制机制创新和深化人才工作机制创新。

学习目标

知识目标

- 理解面向对象编程的基本原则，包括封装、继承和多态（重点：《Python 程序开发职业技能等级标准》初级 1.3.2）。
- 了解类变量和实例变量的区别。
- 理解公有成员和私有成员的区别及其在 Python 中的实现方式。
- 掌握从父类继承属性和方法的方式，并且理解继承的优势（难点：《Python 程序开发职业技能等级标准》初级 1.3.2）。
- 理解代码重用和模块化的原则。

能力目标

- 会使用 Python 定义类，并且创建类的实例。
- 会使用__init__()方法和__del__()方法初始化和清理资源。
- 会定义和使用类的属性和方法，包括 get 方法和 set 方法。
- 会调试和优化面向对象代码。

素质目标

- 通过参与逻辑严密的编程活动，培养自身的逻辑思维能力。
- 在编程实践中，学习并遵循良好的编程习惯和代码规范，建立良好的职业素养。
- 在项目中积极参与团队合作，通过协作完成共同的目标，提高团队协作能力。
- 通过编程学习和实践，激发对编程的浓厚兴趣，树立终身学习的积极态度。

任务分析

本项目以寻宝游戏为设计核心，通过 Python 编程语言，利用面向对象和类的知识，构建一个互动性强、趣味性高的寻宝游戏。在游戏中，玩家将寻找隐藏的宝藏。在项目的实施过程中，学生将深入学习 Python 编程的核心概念，特别是对象和类的理解与运用。学生将学习如何在 Python 中创建和使用对象，定义类及其属性和行为，以此提升编程技能。

要实现的寻宝游戏是在一个虚拟的地图上寻找宝藏。地图在游戏开始时随机生成，包含多个宝藏。玩家需要在地图上移动，寻找宝藏。当玩家找到宝藏时，会获得分数，并且更新战绩板。游戏还包含一个特殊角色——超级玩家，他们拥有一些特殊能力，如双倍移动和查看宝藏位置，使游戏更具趣味性和挑战性。游戏的具体规则如下。

（1）地图生成：游戏开始时，系统会随机生成一个地图，地图上分布着多个宝藏。地图的大小和宝藏的数量可以根据需要进行设置。

（2）玩家角色：玩家在地图上扮演寻宝者的角色。玩家可以选择成为普通玩家或超级玩家。超级玩家拥有一些特殊能力，如双倍移动和查看宝藏位置，这些能力使游戏更具趣味性和挑战性。

（3）玩家移动：玩家可以在地图上上下左右移动，寻找宝藏。每次移动后，系统会检查玩家是否到达了宝藏位置。如果玩家找到了宝藏，则玩家获得分数，并且该宝藏会从地图上消失。

（4）得分和战绩：玩家的得分会实时更新并显示在战绩板上。战绩板会记录所有玩家的得分和排名，以便玩家随时了解自己的成绩，以及与其他玩家的差距。

（5）游戏结束：当所有宝藏都被找到或达到游戏时间限制时，游戏结束。游戏结束时，系统会显示玩家的最终得分和排名。

（6）超级玩家的特殊能力：超级玩家拥有的特殊能力可以在游戏中使用一次。例如，双倍移动能力允许玩家在一次移动中走两步，而查看宝藏位置能力则允许玩家在一段时间内看到地图上所有宝藏的位置。

（7）多人游戏：游戏支持多人同时进行。玩家可以选择与其他玩家合作共同寻找宝藏，或者竞争成为得分最高的寻宝者。

相关知识

面向对象编程（Object-Oriented Programming，OOP）是一种软件设计与编程范式，它运用“对象”的概念来构建软件。在 OOP 中，对象是数据与处理这些数据的方法的结合体。此范式强调数据的封装、继承和多态，旨在提升代码的可复用性、灵活性和扩展性。

OOP 的核心特点如下。

（1）封装性：将数据和操作这些数据的方法封装在类中，仅暴露必要的接口，隐藏内部细节，从而提升安全性和可维护性。

（2）继承性：子类能继承父类的属性和方法，减少代码重复，提高代码的可复用性。

（3）多态性：同一类型的对象能根据不同的情境展现不同的行为，提升代码的灵活性和可扩展性。

（4）抽象性：通过抽象类和接口提炼共性，提高代码的可读性和可维护性。

（5）组合性：将多个对象组合成更复杂的对象，促进代码的复用和扩展。

OOP 的优势如下。

（1）易维护性：使用 OOP 设计的结构清晰，易于阅读和维护，可以降低成本。继承机制使需求变更时只需要进行局部调整。

（2）易扩展性：通过继承减少冗余代码，快速扩展现有功能，提高开发效率。

（3）模块化：封装保护内部数据，提升程序模块化水平，便于后期维护。

（4）可重用性：鼓励模块化和组件化开发，实现代码的高效复用。

（5）可测试性：代码结构便于单元和模块测试，确保质量和可靠性。

（6）方便建模：虽然不完全等同于现实对象，但是 OOP 的对象概念简化了建模过程。

OOP 的特点和优势使其成为复杂软件系统开发的理想选择，有助于开发者更有效地管理和维护代码。同时，OOP 推动更合理的代码组织和更高层次的抽象，对大型项目和团队协作尤为有益。

素质养成

探索精神和创新精神对于国家的发展至关重要。要培养探索精神和创新精神，首先，我们应当保持对未知世界的好奇心，勇于尝试新事物，不畏惧失败和挫折。其次，不断学习新

知识，拓宽自己的知识领域和视野，为创新提供源源不断的灵感和动力。再次，学会独立思考，不盲目跟从，勇于提出自己的见解和想法。最后，付诸实践，将创新想法付诸行动，通过实践来检验和完善自己的想法。同时，社会和国家也应为个人提供宽松的创新环境，鼓励个人勇于创新、敢于探索，为个人的成长和发展提供有力的支持和保障。

本项目通过寻宝游戏弘扬探索精神和创新精神。通过该游戏，学生能深刻体会到探索精神和创新精神的重要性，在游戏中经历解决问题、寻找解决方案的过程，运用逻辑思维和创造力来迎接挑战。项目完成后，学生们不仅能够熟练掌握 Python 编程技能，还能在游戏的实际操作中体验到探索未知事物和勇于尝试的乐趣，从而更加热爱编程和技术创新。

项目实施

本项目将采用 Python 面向对象和类来实现，包含 4 个主要类：地图类（Map）、玩家类（Player）、战绩类（Scoreboard）和超级玩家类（SuperPlayer）。这些类的设计充分考虑了 OOP 的原则，如封装、继承和多态，使代码既易于理解，又便于扩展。

地图类负责生成和显示游戏地图，玩家类处理玩家的移动和交互，战绩类记录和管理玩家的得分，而超级玩家类则是玩家类的子类，拥有特殊能力。这样的设计不仅能清晰分离游戏功能，也提供了一个理解 Python 对象和类概念的实用场景。

任务 1　地图类的实现

地图类是寻宝游戏的核心组件，它负责生成和展示游戏地图，以及管理宝藏的位置。地图类的属性及其含义如表 3.1 所示，地图类的方法及其含义如表 3.2 所示。

微课：项目 3 任务 1-地图类的实现.mp4

表 3.1　地图类的属性及其含义

属性	含义
size	一个元组，用于表示地图的尺寸，如(5, 5)
map	一个二维列表，用于表示地图的布局。每个单元格可以为空（表示没有宝藏）或包含一个宝藏标记
treasure_locations	一个列表，用于存储地图上所有宝藏的位置

表 3.2　地图类的方法及其含义

方法	含义
generate_map()	用于初始化地图的大小和宝藏的位置。为了随机放置宝藏，可以使用 Python 的 random 模块来选择地图上的随机位置。这些位置被添加到 treasure_locations 列表中，并且在 map 二维列表中将相应位置标记为宝藏
display_map()	用于打印地图的当前状态。这个方法遍历地图数组，并且在控制台上显示每个单元格的内容。为了在地图上显示玩家和宝藏，可以在 display_map()方法中添加逻辑来处理这些特殊元素的显示问题

动一动

根据表 3.1 和表 3.2 实现地图类。

任务单

任务单 3-1　地图类的实现

学号：_____　姓名：_____　完成日期：_____　检索号：_____

任务说明

定义地图类，该类应包含生成和显示游戏地图、管理宝藏位置等功能。地图类应能随机生成宝藏，并且在控制台上显示地图状态。此外，地图类应能够在玩家找到宝藏后对宝藏位置进行更新。

引导问题

想一想

（1）什么是面向对象？面向对象和面向过程有什么区别？

（2）定义类的语法是什么？

（3）对类进行实例化的语法是什么？

（4）为什么需要对类进行实例化？

（5）面向对象的类定义能否改为面向过程的？需要注意什么？

重点笔记区

任务评价

评价内容	评价要点	分值	分数评定	自我评价
1. 任务实施	定义地图类	6 分	能正确定义地图类的头部得 1 分；能正确定义属性得 1 分；能正确定义 generate_map()方法得 2 分；能正确定义 display_map()方法得 2 分	
	对类进行实例化	1 分	能正确对类进行实例化得 1 分	
2. 结果展现	运行程序	2 分	能正确打印地图得 2 分	
3. 任务总结	依据任务实施情况进行总结	1 分	总结内容切中本任务的重点和要点得 1 分	
合　计		10 分		

任务解决方案关键步骤参考

（1）选择合适且熟悉的 IDE，创建 Python 脚本文件。导入 random 模块，用于随机生成宝藏放置坐标。

```
import random
```

（2）定义 Map 类。

```
class Map:
```

（3）定义__init__()方法，用于初始化 Map 类的属性，分别为 size、map 和 treasure_locations。其中，size 的初始值在对该类进行实例化时设置，由外部传入。map 的初始值为 size[0]*size[1]的二维列表，所有元素值都被设置为“.”，表示该位置为空。treasure_locations 的初始值为空列表，表示当前未设置任何宝藏。

```
    def __init__(self, size):
```

```
        self.size = size  # 地图的尺寸，如(5, 5)
        self.map = [['.' for _ in range(size[1])] for _ in range(size[0])]
        self.treasure_locations = []  # 宝藏的位置
```

（4）定义 generate_map()方法，该方法首先使用 random 模块随机生成几个宝藏坐标，然后更新地图。将有宝藏的位置的值更改为“T”，表示该位置有宝藏。

```
    def generate_map(self):
        # 随机放置宝藏
        for _ in range(random.randint(1, self.size[0] * self.size[1] // 4)):
            treasure_x = random.randint(0, self.size[1] - 1)
            treasure_y = random.randint(0, self.size[0] - 1)
            self.treasure_locations.append((treasure_x, treasure_y))
            self.map[treasure_x][treasure_y] = 'T'
```

（5）定义 display_map()方法。该方法用于打印地图，对于每个地图位置，“.”表示没有宝藏，“T”表示有宝藏，“P”表示有玩家。

```
    def display_map(self, player_position=None):
        # 显示地图，可以选择显示玩家位置
        for i in range(self.size[0]):
            for j in range(self.size[1]):
                if player_position and (i, j) == player_position:
                    print('P', end=' ')
                else:
                    print(self.map[i][j], end=' ')
            print()  # 换行
```

（6）验证 Map 类的功能。先创建一个大小为 5×5 的 Map 类对象，并且创建一个随机放置宝藏的地图，再调用 display_map()方法打印地图。

```
print("欢迎进入寻宝游戏！")
map = Map((5, 5))
map.generate_map()
map.display_map()
```

（7）保存并运行 Python 脚本文件，运行结果如图 3.1 所示。该程序构造了一个 5×5 的地图，并且随机选取了几个坐标放置宝藏。其中，“.”表示没有宝藏，“T”表示有宝藏。每次运行程序，输出的结果都不同。我们将 map 的二位索引值描述为(x, y)。在打印结果中，左上角即坐标(0, 0)，垂直向下为 x 增长方向，水平向右为 y 增长方向。例如，在图 3.1 中，坐标(1, 1)、(4, 1)、(4, 3)这 3 个位置都放置了宝藏，其他位置都为空。

```
欢迎进入寻宝游戏！
. . . . .
. T . . .
. . . . .
. . . . .
. T . T .
```

图 3.1　Python 脚本文件的运行结果

3.1.1　面向对象

面向对象编程是一种模拟现实世界的程序设计方法。它认为，现实世界由具备各自特性和行为的对象构成。在面向对象编程中，我们将数据和操作这些数据的方法封装成独立的对象，通过操作这些对象实现程序功能。

1. 对象

对象是具体的实体，拥有独特的身份和特征。例如，如图 3.2 所示，可以将狗视为一个对象，其属性包括名字、颜色、年龄等，方法则包括叫、跑、吃等。对象通过类创建，类是定义对象属性和方法的模板。每个基于类创建的对象都有相同的属性和方法，但属性值各异。

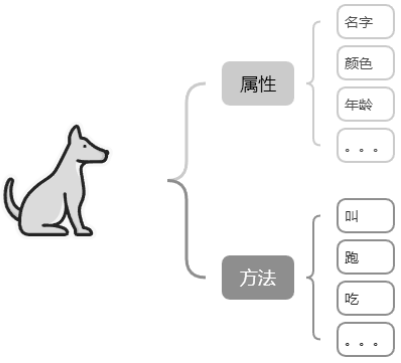

图 3.2　将狗视为一个“对象”

2. 对象的属性和方法

属性是对象的特征或状态，如狗的名字、颜色和年龄。在程序中，属性以变量的形式存储，可以通过对象访问和修改。方法是对象的行为，如狗的叫、跑、吃。在程序中，方法定义在类中，可以通过对象调用。方法可以接收参数并返回结果，调用对象的方法可以使其执行特定的动作。

面向对象编程直观自然，封装数据和操作这些数据的方法形成对象，通过类定义属性和方法，通过对象交互实现程序功能。这种方式提高了代码的可读性和可维护性，使程序更灵活、可扩展。

3. 类

类是面向对象编程的核心。以汽车为例，虽然汽车有轿车、跑车等多种类型，但它们都具备汽车的共同特点。在面向对象编程中，可以将这些类型视为不同的类。类是对具有相似属性和行为事物的抽象描述。属性描述类实例的特性，如汽车的颜色、品牌、型号。方法则是类实例可执行的操作，如汽车的启动、加速、刹车。根据汽车类（如跑车类）的模板可以创建具体实例，即对象。对象具有特定的属性值，可以执行类定义的方法。简而言之，类是模板或蓝图，描述共同的属性和行为；对象是根据类创建的具体实例。

3.1.2　类的定义

在 Python 中，定义类是非常简单的。我们只需要使用 class 关键字，后跟类名即可。以下示例展示了如何定义一个名为 Car 的类。

```
class Car:
    # 类的属性（这里是类变量，通常用于所有实例共享的数据）
    manufacturer = "FAW"

    # 类的初始化方法（构造方法），当创建类的新实例时会自动调用
    def __init__(self, model, color, year):
        # 实例属性（这里是实例变量，每个实例都有自己的值）
        self.model = model
```

```
        self.color = color
        self.year = year

    # 类的方法（实例方法）
    def start_engine(self):
        print(f"Starting the engine of {self.model}...")

    # 另一个类的方法
    def drive(self):
        print(f"Driving {self.color} {self.model} from {self.year}...")
```

在以上示例中，Car 类由 4 部分组成。

（1）类名：Car 是类的名称。

（2）类变量：manufacturer 是一个类变量，它在类的所有实例之间共享。

（3）初始化方法：__init__()方法是一个特殊的方法，当我们创建类的新实例时，Python 会自动调用它。它用于初始化新创建的对象的状态。__init__()方法中的第一个参数总是 self，它引用对象本身，并且允许我们访问或修改对象的属性。

（4）实例方法：start_engine()和 drive()是实例方法，它们定义了在类的实例上可以执行的操作。它们的第一个参数也是 self，代表实例本身。

在上面的代码中，我们并没有创建 Car 类的实例，只是定义了类本身。在实际的程序中，我们可以通过调用类名并传递所需的参数（如果有的话）来创建类的实例。

3.1.3 类的实例化

当定义了一个类之后，我们可以根据这个类来创建实例，这个过程被称为类的实例化。在 Python 中，我们使用类名来调用它以创建一个实例。

以下是使用上面定义的 Car 类来创建实例的示例。

```
# 创建一个 Car 类的实例
my_car = Car("HONGQI", "red", 2021)
# my_car 现在是一个 Car 类的实例（或对象）
# 我们可以访问它的属性
print(my_car.model)  # 输出“HONGQI”
print(my_car.color)  # 输出“red”
print(my_car.year)   # 输出“2021”
# 我们还可以调用它的方法
my_car.start_engine()   # 输出“Starting the engine of HONGQI...”
my_car.drive()          # 输出“Driving red HONGQI from 2021...”
```

在上面的代码中，my_car 是 Car 类的一个实例。我们使用类名 Car 并传递了 3 个参数（车型、颜色和年份）来调用它，从而创建了一个新的实例 my_car。这些参数被传递给__init__()方法，用于初始化新创建的实例的状态。

有了类的实例，我们就可以通过实例来访问其属性和方法。我们使用点运算符和实例名称来访问这些属性和方法。例如，my_car.model 访问 my_car 实例的 model 属性，而 my_car.start_engine()方法则调用 my_car 实例的 start_engine()方法。

每个类的实例都是独立的，它们有自己的状态（实例变量或属性的值），但共享类的相同行为（方法）。封装、继承和多态都建立在类和实例的基础之上。

任务 2　玩家类的实现

玩家类是寻宝游戏中的一个关键组件，负责管理玩家的角色和行为。它包括玩家的属性，如名称和位置，以及允许玩家在地图上移动和寻找宝藏的功能。玩家类通过定义移动方法，让玩家能够根据指令在地图上上下左右移动。此外，玩家类还包含私有方法，用于检查当前位置是否有宝藏。

微课：项目 3 任务 2-玩家类的实现.mp4

玩家类的属性及其含义如表 3.3 所示，玩家类的方法及其含义如表 3.4 所示。

表 3.3　玩家类的属性及其含义

属性	含义
name	玩家的名字，用于标识每个玩家
position	玩家在地图上的位置，表示为坐标点

表 3.4　玩家类的方法及其含义

方法	含义
move(direction, map_size)	根据输入的方向和地图的尺寸，允许玩家在地图上移动
__check_treasure(self, map)	用于检查玩家当前位置是否有宝藏

动一动

根据表 3.3 和表 3.4 实现玩家类。

任务单

任务单 3-2　玩家类的实现

学号：_____　姓名：_____　完成日期：_____　检索号：_____

任务说明

定义玩家类，该类应包含管理玩家角色和行为的功能。玩家类包含玩家的名称和位置属性，以及允许玩家在地图上移动。同时，玩家类包含一个私有方法，用于检查当前位置是否有宝藏。

引导问题

想一想

（1）什么是类成员？列举一些类成员的类型。

（2）如何在 Python 中定义类成员？

（3）什么是构造函数？什么是析构函数？

（4）在 Python 中，构造函数和析构函数的特殊方法名称是什么？

（5）构造函数和析构函数有什么不同之处？

（6）什么是公有函数？什么是私有函数？

（7）如何在类中定义公有函数和私有函数？

重点笔记区

续表

任务评价

评价内容	评价要点	分值	分数评定	自我评价
1. 任务实施	定义玩家类	5 分	能正确定义__init__()方法得 1 分；能正确定义 move()方法得 2 分；能正确定义__check_treasure()方法得 2 分	
	创建玩家类对象	1 分	能正确创建玩家类对象得 1 分	
	移动玩家并检查宝藏	2 分	能正确调用 move()方法进行移动得 1 分；能正确调用__check_treasure()方法检查宝藏得 1 分	
2. 结果展现	运行程序	1 分	程序能正常执行，并且正常输出结果得 1 分	
3. 任务总结	依据任务实施情况进行总结	1 分	总结内容切中本任务的重点和要点得 1 分	
合　计		10 分		

任务解决方案关键步骤参考

（1）打开任务 3-1 的 Python 脚本文件，继续编辑代码，定义 Player 类。

```
class Player:
```

（2）定义 Player 类的__init__()方法，初始化 name 属性和 position 属性。其中，name 表示玩家的名字，为字符串，position 表示玩家的初始位置，为一个元组。两个属性的初始值都由外部传入。

```
    def __init__(self, name, position):
        self.name = name  # 玩家的名字
        self.position = position  # 玩家的初始位置，格式为(x, y)
```

（3）定义 move()方法，该方法向指定的方向前进一步，进而更新 position 属性的值为新位置的坐标。该方法需要检查指定方向是否可行进。如果指定方向已经到头了，则不更新位置。

```
    def move(self, direction, map_size):
        # 根据方向移动玩家
        x, y = self.position
        if direction == 'up' and x > 0:
            x -= 1
        elif direction == 'down' and x < map_size[0] - 1:
            x += 1
        elif direction == 'left' and y > 0:
            y -= 1
        elif direction == 'right' and y < map_size[1] - 1:
            y += 1
        self.position = (x, y)
```

（4）定义__check_treasure()方法。该方法检查玩家当前所在位置是否有宝藏。如果有，则打印“你发现了宝藏!”，并且取走该位置的宝藏，返回 True；否则，返回 False。取走宝藏的操作就是将地图中该位置的值由“T”变更为“.”。

```
    def __check_treasure(self, map):
        # 检查玩家当前所在位置是否有宝藏
        x, y = self.position
        if map.map[x][y] == 'T':
            print("你发现了宝藏！")
```

```
            map.map[x][y] = '.'  # 清除宝藏
            return True
        return False
def call_check(sefl, map):
        return self.__check_treasure(map)
```

（5）验证 Player 类的功能。创建一个大小为 5×5 的 Map 类对象，并且创建一个随机放置宝藏的地图。

```
print("欢迎进入寻宝游戏！")
map = Map((5, 5))
map.generate_map()
```

（6）创建 Player 类对象，将玩家的名字设置为“陈杰”，初始位置设置为(0, 0)。

```
player = Player("陈杰", (0, 0))
```

（7）调用 display_map()方法，传入玩家的位置，通过输出信息查看当前的地图状态，包括宝藏位置和玩家位置。其中，宝藏使用“T”表示，玩家使用“P”表示。

```
map.display_map(player.position)
```

（8）首先调用 move()方法进行移动，然后再次调用 display_map()方法显示当前的地图状态，调用 call_check()方法检查当前位置是否有宝藏。

```
# 玩家移动并检查宝藏
print("移动玩家位置...")
player.move('right', map.size)
map.display_map(player.position)
player.call_check(map)
```

（9）保存 Python 脚本文件并运行，运行效果如图 3.3 所示。程序首先创建一个 5×5 的地图，并且随机放置宝藏。每次运行程序，放置宝藏的结果都会不同。在本次运行过程中，程序在两个位置放置了宝藏，分别是(1, 3)和(0, 2)。此外，玩家位于地图(0, 0)处。地图其他位置皆为空。随后，玩家向右移动了一个位置，因此更新位置到(0, 1)。此时，玩家未发现宝藏，因此并未输出“你发现了宝藏!”，宝藏的位置也没有发生任何变化。

```
欢迎进入寻宝游戏!
P . . . .
. . . T .
T . . . .
. . . . .
. . . . .
移动玩家位置...
. P . . .
. . . T .
T . . . .
. . . . .
. . . . .
```

图 3.3　Python 脚本文件的运行效果

3.2.1　类的成员

在面向对象编程中，类的成员对于定义类的特质和行为至关重要，它们使类的实例（对象）具备独特的状态和功能。类的成员主要包括两种类型：属性和方法。

1. 实例属性和类属性

实例属性也被称为对象属性，这些属性隶属于各个对象，每个对象都拥有独立的实例属性值。它们通常通过__init__()方法进行初始化，并且只能通过对象来访问。以 Car 类为例，model、color 和 year 就是实例属性。

类属性（静态属性）属于类本身，所有对象共享。它们可以在类外部直接通过类名访问，或者在类内部通过类名或对象访问。在 Car 类中，manufacturer 就是一个类属性。

2. 实例方法、类方法和静态方法

实例方法是类中最常见的方法类型，通过对象调用。它们能访问并修改对象的实例属性，并且通常以 self 作为第一个参数，代表对象本身。例如，Car 类中的 start_engine()方法和 drive()方法就是实例方法。

类方法通过@classmethod 装饰器定义，可以通过类或对象调用。类方法通常以 cls 作为第一个参数，代表类本身，主要用于操作类级别的属性或执行与类直接相关的操作。

静态方法通过@staticmethod 装饰器定义，同样可以通过类或对象调用。但静态方法与类及对象无特殊关联，更像一般的方法，只是位于类的命名空间中。

此外，Python 类还支持__init__()（构造方法）、__str__()（定义对象字符串的表现形式）、__call__()（使对象可以像函数那样被调用）等特殊方法。这些方法多用于定义对象的行为，如初始化、类型转换或比较等。

通过精心定义类的成员，我们能够创建出结构明晰、功能丰富的类，从而充分发挥面向对象编程的优势。

3.2.2 构造函数和析构函数

在面向对象编程中，构造函数（Constructor）和析构函数（Destructor）在对象的生命周期中扮演着重要的角色。在 Python 中，并没有显式的析构函数，但可以通过特殊的方法（如__init__()方法和__del__()方法）来实现类似的功能。

1. 构造函数

构造函数用于在创建新对象时初始化对象的状态。在大多数面向对象的编程语言中，当使用 new 关键字（或其等价物）创建对象时，构造函数会自动被调用。

在 Python 中，构造函数是通过__init__()方法实现的。__init__()方法是一个特殊的方法，在创建类的实例时自动调用。它允许我们为新创建的对象设置初始状态。

```
class MyClass:
    def __init__(self, value):
        self.value = value

# 创建一个 MyClass 实例，并且传入一个值来初始化它
obj = MyClass(10)
print(obj.value)  # 输出"10"
```

在上面的例子中，__init__()方法接收一个 value 参数，并且将其设置为新对象的 value 属性。

2. 析构函数

析构函数在对象被销毁之前自动调用，它允许我们执行一些清理工作，如释放资源、关闭文件等。在 Python 中，可以通过定义__del__()方法来实现类似的功能。__del__()方法在对象被垃圾回收器回收之前调用，但请注意，这个调用并不是确定的，因为 Python 的垃圾回收器是自动的，并且可能在任何时候运行。

```
class MyClass:
    def __init__(self, value):
```

```
        self.value = value

    def __del__(self):
        print("Object is being destroyed")

# 创建一个 MyClass 实例
obj = MyClass(10)
```

当 obj 没有其他引用时，它可能会被垃圾回收器回收。此时，__del__()方法可能会被调用。然而这里的输出可能并不会立即显示，因为它依赖于垃圾回收器的行为。

由于 Python 的垃圾回收器是自动的，并且不受开发者的直接控制，因此__del__()方法通常不是执行关键清理任务的理想选择。对于需要明确释放的资源（如文件句柄、数据库连接等），更好的做法是使用上下文管理器（通过实现__enter__()方法和__exit__()方法）或 with 语句，或者使用 try/finally 块来确保资源被释放。

3.2.3　类成员修饰符

在 Python 中，类成员修饰符主要用于控制访问权限。Python 不像一些其他语言（如 C++、Java）那样提供明确的访问修饰符关键字（如 public、private、protected），而是通过命名约定和一些特殊方法来实现访问控制。

公开（public）成员可以被类的实例、子类及外部代码直接访问。通常，名称以除单下划线（_）或双下划线（__）开头外的成员被视为公开成员。

保护（protected）成员意在告知使用者这些变量是供内部使用的，不建议在类的外部或子类中直接访问。但是，在技术上仍然可以直接访问。通常，名称以单下划线（_）开头的成员被视为保护成员（这只是一种命名约定，Python 实际上没有保护成员的概念）。

私有（private）成员不能直接从类的外部访问，但可以通过类内部的特定方法访问。私有成员主要用于封装类内部的实现细节，防止外部干扰。通常，名称以双下划线（__）开头的成员会被 Python 解释器进行名称改写（Name Mangling），以此实现一定程度的私有性。

3.2.4　私有函数

在大多数编程语言中，没有特定的语法来显式地声明一个函数是公有的，除非使用特定的语法来声明函数是私有的或受保护的。在 Python 中，所有没有特殊前缀的函数都被认为是公有的。例如：

```
class MyClass:
    def public_method(self):
        print("This is a public method.")

# 创建实例并调用公有函数
obj = MyClass()
obj.public_method()  # 输出“ This is a public method.”
```

在以上代码中，public_method()是一个公有函数，可以在类的外部通过实例 obj 来调用。

私有函数（Private Functions）是在类的外部不能直接访问的函数。它们主要用于类的内部实现细节，并且通常不希望在类的外部被调用。不同的编程语言使用不同的方式来定义私有函数。

在 Python 中，虽然没有严格意义上的私有函数，但有一种约定俗成的做法是使用单下划线（_）作为方法名的前缀来表示该函数是“受保护的”或“内部使用的”，或者使用双下划线（__）作为前缀来表示该函数是“私有的”并触发名称改写。

使用单下划线的示例如下。

```
class MyClass:
    def _protected_method(self):
        print("This is a protected method.")

# 创建实例并尝试调用受保护的函数（在 Python 中仍然是可访问的）
obj = MyClass()
obj._protected_method()  # 输出: This is a protected method.
```

使用双下划线的示例如下。

```
class MyClass:
    def __private_method(self):
        print("This is a private method.")

    def call_private_method(self):
        self.__private_method()

# 创建实例并尝试直接调用私有函数（在 Python 中会引发错误）
obj = MyClass()
# obj.__private_method()  # 这会引发 AttributeError

# 但可以通过类中的其他公有函数间接调用私有方法
obj.call_private_method()  # 输出“This is a private method.”
```

在以上示例中，尽管__private_method()被设计为私有函数，但它仍然可以通过类中的其他公有函数（如 call_private_method()）来间接调用。

公有函数和私有函数的概念是面向对象编程中封装性的一个重要部分，允许我们控制对类内部的访问，从而保护数据并隐藏不必要的复杂性。

任务 3　战绩类的实现

微课：项目 3 任务 3-战绩类的实现.mp4

战绩类负责记录和管理所有玩家的得分和排名。它存储玩家的得分信息，并且在游戏过程中实时更新这些信息，以显示玩家的得分和排名。通过战绩类，玩家可以随时了解自己的得分和与其他玩家的差距，增加游戏的竞争性和互动性。战绩类需要存储和显示玩家的得分，并且在游戏过程中实时更新玩家得分。

战绩类的属性及其含义如表 3.5 所示，战绩类的方法及其含义如表 3.6 所示。

表 3.5　战绩类的属性及其含义

属性	含义
players	一个字典，存储所有玩家的名字及其得分

表 3.6　战绩类的方法及其含义

方法	含义
add_player(player)	添加一个新玩家到战绩板
update_score(player, points)	更新某个玩家的得分
get_score(player)	获取某个玩家的得分
set_score(player, score)	设置某个玩家的得分
display_scores()	显示所有玩家的名字和得分

动一动

根据表 3.5 和表 3.6 实现战绩类。

任务单

任务单 3-3　战绩类的实现

学号：_____　姓名：_____　完成日期：_____　检索号：_____

任务说明

定义战绩类，该类负责记录和管理所有玩家的得分和排名。战绩类应包含玩家的得分信息存储、得分更新、得分显示等功能，以实时展示玩家的名字和得分。

引导问题

想一想

（1）什么是类属性？什么是对象属性？
（2）类属性和对象属性之间有什么区别？
（3）什么是公有属性？什么是私有属性？
（4）在 Python 中，如何定义公有属性和私有属性？
（5）请解释私有属性的命名约定。
（6）什么是 get 方法？什么是 set 方法？
（7）为什么需要使用 get 方法和 set 方法？

重点笔记区

任务评价

评价内容	评价要点	分值	分数评定	自我评价
1. 任务实施	定义战绩类	5 分	能正确定义类属性得 1 分；能正确定义私有属性得 1 分；能正确定义 get 方法和 set 方法得 3 分	
	创建战绩对象	1 分	能正确创建战绩对象得 1 分	
	对战绩对象进行操作	2 分	能正确调用战绩对象的方法得 2 分	
2. 结果展现	运行程序	1 分	程序能正常执行，并且正确输出玩家名字和得分信息得 1 分	
3. 任务总结	依据任务实施情况进行总结	1 分	总结内容切中本任务的重点和要点得 1 分	
合　计		10 分		

任务解决方案关键步骤参考

（1）选择合适且熟悉的 IDE，打开任务 3-2 的 Python 脚本文件，定义 Scoreboard 类。

```
class Scoreboard:
```

（2）定义类属性 players，用于存储游戏中的所有玩家对象。在游戏中，我们可以创建多个计分面板，每个计分面板都只管理自己的玩家的得分。但所有计分面板的所有玩家对象都会被添加到类属性 players 中。

```
    players = {}  # 类属性，存储所有玩家对象
```

（3）定义__init__()方法。定义__player_scores 属性，该属性是当前积分面板所管理的玩家名字和得分的键值对字典。将__player_scores 属性的初始值设置为空字典。

```
    def __init__(self):
        self.__player_scores = {}  # 私有属性，存储当前积分面板的玩家名字和得分
```

（4）定义 add_player()方法。该方法将一个玩家对象添加到 players 字典中，将玩家名字添加到__player_scores 字典中，并且将其初始分值设置为 0 分。

```
    def add_player(self, player):
        self.players[player.name] = player
        self.__player_scores[player.name] = 0
```

（5）定义 update_score()方法。该方法为指定玩家增加指定分数。

```
    def update_score(self, player, points):
        self.__player_scores[player.name] += points
```

（6）定义 get_score()方法。该方法获取指定玩家的得分并返回。如果玩家对象未被添加到 players 字典中，则返回 0。

```
    def get_score(self, player):
        return self.__player_scores.get(player.name, 0)
```

（7）定义 set_score()方法。该方法为指定玩家直接设置指定分数。

```
    def set_score(self, player, score):
        self.__player_scores[player.name] = score
```

（8）定义 display_scores()方法。该方法遍历__player_scores 字典，打印当前所有玩家的名字和得分。

```
    def display_scores(self):
        for name, score in self.__player_scores.items():
            print(f"{name}: {score}")
```

（9）验证 Scoreboard 类的功能。首先创建 Scoreboard 类对象。

```
print("欢迎进入寻宝游戏！")
scoreboard = Scoreboard()
```

（10）创建两个玩家对象，一个玩家对象名为“陈杰”，初始位置为(0, 0)；另一个玩家对象名为“丽丽”，初始位置为(0, 1)。

```
player1 = Player("陈杰", (0, 0))
player2 = Player("丽丽", (0, 1))
```

（11）调用 add_player()方法，将这两个玩家对象添加到 players 字典中。

```
scoreboard.add_player(player1)
scoreboard.add_player(player2)
```

（12）调用 update_score()方法更新玩家的得分。该方法为指定玩家增加指定分数。起初，所有玩家的初始分数都是 0 分。将“陈杰”的得分更新为 10 分，将“丽丽”的得分更新为 5 分。

```
scoreboard.update_score(player1, 10)
scoreboard.update_score(player2, 5)
```

（13）调用 display_scores()方法，打印当前所有玩家的名字和得分。

```
scoreboard.display_scores()
```

（14）保存 Python 脚本文件并运行。当前程序添加了两名玩家，分别是“陈杰”和“丽丽”，其得分分别为 10 分和 5 分。display_scores()方法会遍历__player_scores 字典，并且打印所有键值对，最终输出结果如图 3.4 所示。

```
欢迎进入寻宝游戏!
陈杰: 10
丽丽: 5
```

图 3.4　最终输出结果

3.3.1　公有属性和私有属性

公有属性（Public Attributes）和私有属性（Private Attributes）在面向对象编程中主要用于封装类的内部状态，并且控制对类数据的访问。

1. 公有属性

公有属性是可以在类的外部直接访问的属性。在 Python 中，我们可以直接在类定义中设置公有属性，并且通过对象来访问它们。

```
class MyClass:
    def __init__(self, public_attr):
        self.public_attr = public_attr

# 创建实例并设置公有属性
obj = MyClass(public_attr="Hello, World!")

# 访问公有属性
print(obj.public_attr)  # 输出“Hello, World!”

# 修改公有属性
obj.public_attr = "Goodbye, World!"
print(obj.public_attr)  # 输出“Goodbye, World!”
```

在以上代码中，public_attr 是一个公有属性，它可以在类的外部被创建、访问和修改。

2. 私有属性

私有属性是在类的外部不能直接访问的属性。它们主要用于隐藏类的内部实现细节，防止外部代码错误地修改它们，从而保持对象的封装性。在 Python 中，通常使用单下划线（_）作为前缀来表示一个属性是“受保护的”或“内部使用的”，但这并不是真正的私有属性，只是一种约定。然而，为了模拟私有属性，Python 有一个惯例，即使用双下划线（__）作为前缀，这会导致属性名称被“修饰”（mangled），从而阻止从外部直接访问。

下面是一个使用双下划线前缀的示例。

```
class MyClass:
    def __init__(self, private_attr):
        self.__private_attr = private_attr

    def get_private_attr(self):
```

```
        return self.__private_attr

    def set_private_attr(self, value):
        self.__private_attr = value

# 创建实例并设置私有属性（通过 setter()方法）
obj = MyClass(private_attr="Secret")

# 访问私有属性（通过 getter()方法）
print(obj.get_private_attr())  # 输出“Secret”

# 不能直接修改私有属性（但可以通过 setter()方法修改）
# obj.__private_attr = "Not so secret"  会引发 AttributeError
obj.set_private_attr("Not so secret")
print(obj.get_private_attr())  # 输出“Not so secret”
```

在以上示例中，__private_attr 是一个私有属性，它不能在类的外部直接访问或修改。相反，我们使用了两个公有方法 get_private_attr()和 set_private_attr()来间接地访问和修改私有属性的值。这样，我们就可以控制对私有属性的访问，并且在必要时执行一些额外的逻辑（如验证输入或触发其他事件）。

需要注意的是，即使使用了双下划线前缀，私有属性仍然可以在类的内部直接访问和修改，甚至可以通过特殊的名称修饰规则从类的外部间接访问（但这通常是不推荐的）。因此，私有属性应该仅用于封装类的内部实现细节，防止外部代码错误地修改它们。

3.3.2 get 方法和 set 方法

get 方法和 set 方法（通常也被称为 getter 方法和 setter 方法）在面向对象编程中常用于封装对象的属性，以便控制对属性的访问和修改。这些方法提供了一种机制，允许我们在获取或设置属性值之前或之后执行额外的逻辑，如验证输入、触发事件或更新其他属性。

1. get 方法

get 方法用于获取属性的值。在 Python 中，通常没有特定的语法来定义 get 方法，但我们可以通过定义一个返回属性值的普通方法来模拟它。get 方法通常被命名为“get_<属性名>”（尽管这不是强制的）。

```
class MyClass:
    def __init__(self, value):
        self._private_value = value

    def get_value(self):
        # 这里可以添加额外的逻辑，但在简单的示例中我们直接返回属性值
        return self._private_value

# 创建实例并访问属性值
obj = MyClass(10)
print(obj.get_value())  # 输出“10”
```

在以上代码中，_private_value 是一个“私有”属性（尽管在 Python 中，单下划线前缀只是约定俗成的表示方法，并不是真正的私有），而 get_value()方法是一个 get 方法，用于获取

_private_value 属性的值。

2. set 方法

set 方法用于设置属性的值。与 get 方法类似，set 方法也没有特定的语法，但通常命名为“set_<属性名>”（尽管这不是强制的）。set 方法应该接收一个参数（要设置的新值），并且可能包含一些额外的逻辑，如验证输入或更新其他属性。

```
class MyClass:
    def __init__(self, value):
        self._private_value = value

    def get_value(self):
        return self._private_value

    def set_value(self, new_value):
        # 这里可以添加额外的逻辑，如验证输入值
        if isinstance(new_value, int) and new_value >= 0:
            self._private_value = new_value
        else:
            raise ValueError("Value must be a non-negative integer.")

# 创建实例并设置属性值
obj = MyClass(10)
print(obj.get_value())  # 输出“10”

# 尝试设置一个新值
obj.set_value(20)
print(obj.get_value())  # 输出“20”

# 尝试设置一个无效的值（将引发异常）
obj.set_value(-1)  # ValueError: Value must be a non-negative integer.
```

在以上代码中，set_value()方法是一个 set 方法，它接收一个 new_value 参数，并且验证该参数是否是一个非负整数。如果 new_value 参数是有效的，则将其设置为_private_value 属性的新值；否则，将引发 ValueError 异常。

3. @property 装饰器

在 Python 中，我们可以使用@property 装饰器来定义 get 方法，并且使用“@<属性名>.setter 装饰器”来定义 set 方法。这样做可以使属性访问看起来更像普通的属性访问，而不是方法调用。

```
class MyClass:
    def __init__(self, value):
        self._private_value = value

    @property
    def value(self):
        return self._private_value

    @value.setter
    def value(self, new_value):
```

```
        if isinstance(new_value, int) and new_value >= 0:
            self._private_value = new_value
        else:
            raise ValueError("Value must be a non-negative integer.")

# 创建实例并访问/设置属性值
obj = MyClass(10)
print(obj.value)   # 输出“10”
obj.value = 20     # 使用 setter 方法设置属性值
print(obj.value)   # 输出“20”
obj.value = -1     # ValueError: Value must be a non-negative integer.
```

在以上代码中，我们使用了@property 装饰器和@value.setter 装饰器来定义 value 属性的 get 方法和 set 方法。这使我们可以像访问普通属性一样来访问和设置 value 属性，但实际上是在调用背后定义的 get 方法和 set 方法。

任务 4　超级玩家类的实现

微课：项目 3 任务 4-超级玩家类的实现.mp4

超级玩家类是寻宝游戏中的一个特殊角色，旨在增加游戏的趣味性和挑战性。它继承玩家类，但拥有额外的特殊能力，如双倍移动和查看宝藏位置。超级玩家类使玩家能够以更独特的方式探索地图，找到宝藏，增加游戏的竞争性和互动性。

超级玩家的双倍移动能力允许该玩家在一次移动中走两步，而普通玩家只能走一步。这使超级玩家在探索地图时能够更快地到达宝藏位置。超级玩家的查看宝藏位置能力允许该玩家查看地图上所有宝藏的确切位置，而不仅仅是他们当前所在位置附近的宝藏。这种能力使超级玩家在寻宝过程中能够做出更有效的决策，提高找到宝藏的概率。

超级玩家类的属性及其含义如表 3.7 所示，超级玩家类的方法及其含义如表 3.8 所示。

表 3.7　超级玩家类的属性及其含义

来源	属性	含义
继承父类	name	超级玩家的名字，用于标识角色
继承父类	position	超级玩家在地图上的位置，表示为坐标点
子类	special_power	一个字符串，表示超级玩家的特殊能力，如 double_move 表示双倍移动，see_treasure 表示查看宝藏位置

表 3.8　超级玩家类的方法及其含义

来源	方法	含义
继承父类	move(direction, map_size)	允许超级玩家在地图上移动。与普通玩家类的方法相同，但可以实现双倍移动的特殊逻辑
子类	use_special_power(self)	一个特殊方法，用于使用超级玩家的特殊能力。根据 special_power 的值，实现不同的特殊能力逻辑

动一动

根据表 3.7 和表 3.8 实现超级玩家类。

任务单

任务单 3-4　超级玩家类的实现

学号：_____　姓名：_____　完成日期：_____　检索号：_____

任务说明

定义超级玩家类，该类继承玩家类，并且包含双倍移动和查看宝藏位置的特殊能力。超级玩家类应包含玩家的属性，如名称和位置，以及使用特殊能力的方法。同时，该类应能够实现特殊能力的逻辑，超级玩家能够使用双倍移动和查看宝藏位置的能力。

引导问题

想一想

（1）什么是类的继承？为什么需要使用继承？

（2）如何在 Python 中创建一个子类并继承父类的属性和方法？

（3）类的属性是否可以被子类继承？如果可以，如何实现属性的继承？

（4）子类是否可以拥有自己的属性？

（5）类的方法是否可以被子类继承？如果可以，如何实现方法的继承？

重点笔记区

任务评价

评价内容	评价要点	分值	分数评定	自我评价
1．任务实施	定义超级玩家类	5 分	能正确定义超级玩家类，并且继承玩家类得 1 分；能正确定义__init__()方法得 1 分；能正确定义 use_special_power()方法得 3 分	
	创建超级玩家对象	2 分	能正确创建超级玩家对象得 2 分	
	使用特殊能力	1 分	能正确使用特殊能力得 1 分	
2．结果展现	运行程序	1 分	程序能正常执行，并且正确打印地图状态得 1 分	
3．任务总结	依据任务实施情况进行总结	1 分	总结内容切中本任务的重点和要点得 1 分	
合　计		10 分		

任务解决方案关键步骤参考

（1）选择合适且熟悉的 IDE，打开任务 3-3 的 Python 脚本文件，修改 Player 类，以支持子类的双倍移动能力。修改 Player 类的 move()方法，接收 double_move 参数，表示当前是否为双倍移动。将该参数的值设置为 False。

```
    def move(self, direction, map_size, double_move=False):
```

（2）在 move()方法中添加双倍移动的逻辑。对于每个方向的移动，默认走 1 步，如果当前使用了双倍移动能力，则走两步。

```
        # 根据方向移动玩家
        x, y = self.position
        if direction == 'up' and x > 0:
            x -= 1
```

```
            if double_move and x > 0:  # 双倍移动
                x -= 1
        elif direction == 'down' and x < map_size[0] - 1:
            x += 1
            if double_move and x < map_size[0] - 1:  # 双倍移动
                x += 1
        elif direction == 'left' and y > 0:
            y -= 1
            if double_move and y > 0:  # 双倍移动
                y -= 1
        elif direction == 'right' and y < map_size[1] - 1:
            y += 1
            if double_move and y < map_size[1] - 1:  # 双倍移动
                y += 1
        self.position = (x, y)
```

（3）定义 SuperPlayer 类，该类继承 Player 类。

```
class SuperPlayer(Player):
```

（4）定义__init__()方法，其中，首先调用父类的__init__()方法，然后定义子类的 special_power 属性。当前，该属性的有效值为 double_move 或 see_treasure。该属性的初始值从外部传入。

```
    def __init__(self, name, position, special_power):
        super().__init__(name, position)
        self.special_power = special_power
```

（5）定义 use_special_power()方法。该方法根据不同的 special_power 属性执行不同的代码逻辑。对于 double_move，调用 move()方法，并且将 double_move 参数的值设置为 True；对于 see_treasure，遍历地图中的每个位置，检查遍历到的位置是否有宝藏，并且输出有宝藏的坐标。如果 special_power 属性既不是 double_move 也不是 see_treasure，则输出“未知技能！”

```
    def use_special_power(self, direction, map):
        if self.special_power == "double_move":
            print("使用超级技能：双倍移动！")
            self.move(direction, map.size, double_move=True)  # 示例：向上双倍移动
        elif self.special_power == "see_treasure":
            print("使用超级技能：查看宝藏位置！")
            # 显示地图上的所有宝藏位置
            for x, y in map.treasure_locations:
                print(f"发现宝藏：({x}, {y})")
        else:
            print("未知技能！")
```

（6）验证 SuperPlayer 类的功能。首先创建 5×5 的地图对象，并且随机生成宝藏。

```
print("欢迎进入寻宝游戏！")
map = Map((5, 5))
map.generate_map()
```

（7）创建 SuperPlayer 类的对象，将名字设置为“超级陈杰”，初始位置为(0, 0)，特殊能力为 double_move。

```
super_player = SuperPlayer("超级陈杰", (0, 0), "double_move")
```

（8）调用 use_special_power()方法，触发超级玩家对象的特殊能力。

```
super_player.use_special_power("right", map)
```

（9）调用 display_map()方法，查看当前的地图状态。

```
map.display_map(super_player.position)
```

（10）保存并运行 Python 脚本文件，运行结果如图 3.5 所示。该程序构建了一个 5×5 的地图，并且随机放置了宝藏。在本次运行过程中，程序只在(0, 3)位置放置了宝藏。创建超级玩家“超级陈杰”，将初始位置设置为(0, 0)，并且赋予了他双倍移动能力。“超级陈杰”从其初始位置(0, 0)开始向右移动，并且使用了双倍移动能力，所以他会向右移动两步。因此，他的最终位置为(0, 2)。

```
欢迎进入寻宝游戏!
使用超级技能：双倍移动!
. . P . .
. . . . .
. . . . .
T . . . .
. . . . .
```

图 3.5　Python 脚本文件的运行结果

3.4.1　父类和子类

父类（基类或超类）和子类（派生类）是面向对象编程中两个重要的概念，它们用于实现代码的复用和扩展性。子类继承父类，并且可以添加新的特性或修改从父类继承的特性。

父类是一个被其他类继承的类。它定义了一些通用的属性和方法，这些属性和方法可以被其子类共享和重用。子类可以继承父类的所有属性和方法（除非它们在子类中被覆盖或隐藏），并且可以添加新的属性和方法。子类也可以覆盖父类的方法，以便提供特定的实现。

例如，Animal 是父类，它定义了一个动物的基本属性和行为（如名字和发声）。Dog 类是子类，它继承了 Animal 类的所有属性和方法，并且可以添加新的属性（如品种）和新的方法（如摇尾巴）。此外，Dog 类还可以覆盖 Animal 类相应的方法，如提供狗特有的发声方式。

super().__init__(name)调用是 Python 中的一种常见模式，这种模式用于在子类的构造函数中调用父类的构造函数，以确保父类中的初始化代码被正确执行。如果不这样做，子类可能无法正确初始化从父类继承的属性。

通过使用父类和子类，我们可以构建出更加模块化和可复用的代码，并且轻松地扩展和修改对象的行为。

3.4.2　属性的继承

在面向对象编程中，子类能够自动继承其父类定义的属性，这被称为属性的继承。当创建子类实例时，它将同时包含自己定义的属性和从父类继承的属性。

父类中未使用特殊前缀（如__）的属性被视为公有属性，子类可以直接访问和继承这些属性。以单下划线（_）开头的属性通常被视为保护属性，而以双下划线（__）开头的属性则会被 Python 解释器改变名字，变为私有属性。尽管这些保护属性和私有属性可以被继承，但不建议在子类外部直接访问。

值得注意的是，在 Python 中不存在真正的私有属性，因为所有属性均可通过类方法或对象的__dict__属性进行访问。

3.4.3 方法的继承

在面向对象编程中，方法的继承是指子类自动获得其父类定义的方法。这意味着当我们创建一个子类的实例时，它不仅可以使用自己定义的方法，还可以使用从父类继承的方法。当子类继承父类时，它会继承父类中定义的所有公有方法（没有双下划线前缀的方法）。这些公有方法可以直接被子类的实例调用。如果子类需要修改父类方法的行为，则可以在子类中定义同名的方法，这被称为方法的重写（Overriding）。下面是一个简单的示例。

```
# 父类 Animal
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound")

# 子类 Dog
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)    # 调用父类的构造函数，继承 name 属性
        self.breed = breed        # 子类特有的属性

    # 重写父类的 speak()方法
    def speak(self):
        print(f"{self.name} barks")

    def wag_tail(self):
        print(f"{self.name} wags its tail")

# 创建子类的实例
my_dog = Dog("Buddy", "Golden Retriever")

# 调用继承父类的方法
my_dog.speak()  # 输出“Buddy barks”（调用的是子类重写的 speak()方法）

# 调用子类特有的方法
my_dog.wag_tail()  # 输出“Buddy wags its tail”

# 假设我们有一个 Animal 实例
my_animal = Animal("Generic Animal")
my_animal.speak()  # 输出“Generic Animal makes a sound”（调用的是父类的 speak()方法）
```

在以上示例中，Dog 类继承了 Animal 类的 speak()方法。但是，Dog 类重写了 speak()方法，提供了狗特有的发声方式（吠叫）。当我们创建 Dog 实例并调用 speak()方法时，会执行子类重写的 speak()方法。而当我们创建 Animal 实例并调用 speak()方法时，会执行父类中定义的 speak()方法。此外，Dog 类还定义了一个特有的方法 wag_tail()，这个方法只能被 Dog 实

例调用。

通过方法的继承，我们可以实现代码的复用和扩展，使子类在保持与父类相似行为的同时，能够添加自己特有的功能。

拓展实训：飞机大战游戏的实现

【实训目的】

通过本拓展实训，学生能够掌握 Python 面向对象的编程技巧，包括类的定义和实例化、类的成员属性和方法、构造函数和析构函数、公有函数和私有函数、类属性和对象属性及类的继承。

【实训环境】

Python IDLE、Python 3.11。

【实训内容】

飞机大战游戏是一种经典的射击游戏，通常在二维空间中进行。玩家控制一架飞机，目标是尽可能长时间地存活，同时摧毁尽可能多的敌机来获得分数。这个游戏具有以下特点。

（1）游戏中会有不同类型的敌机，它们可能具有不同的移动模式、攻击方式和生命值。

（2）玩家可以通过摧毁敌机或特定目标来获得升级道具，这些道具可以增强玩家飞机的能力，如提升火力、速度或添加额外生命。

（3）游戏可能包含多个关卡，每个关卡都有其独特的敌机波次和难度设置。

（4）玩家通过摧毁敌机来获得分数，分数会根据敌机的类型和难度而有所不同。

本实训将通过 Python 面向对象的编程思想设计与实现一个飞机大战游戏。目前，我们仅考虑后端功能逻辑的实现，并且对完整游戏规则进行了简化。

游戏规则如下。

（1）玩家需要控制飞机的移动，避开敌机的攻击。

（2）玩家飞机可以通过发射子弹来摧毁敌机。

（3）玩家飞机有一定的生命值，被敌机击中或撞击到敌机会减少生命值。当生命值耗尽时，游戏结束。

（4）摧毁敌机可以获得分数，积累一定的分数可以增强飞机的能力。

（5）当玩家飞机的生命值耗尽时，游戏结束。玩家可以记录分数并尝试在下一次游戏中打破纪录。

（6）游戏采用回合制，所有飞机轮流执行一个回合。玩家或敌机在它掌握的回合中，选择移动或攻击其中一个动作。

本实训实现了玩家飞机的移动、攻击和敌机的移动，未实现敌机的攻击。根据以上对于游戏规则的描述，需要设计以下几个类。

（1）Plane 类（基础飞机类）：游戏中所有飞机的父类，它代表了游戏中所有飞机的基本功能和属性，如移动位置和发射子弹。Plane 类的属性及其含义如表 3.9 所示，Plane 类的方法及其含义如表 3.10 所示。

表 3.9　Plane 类的属性及其含义

属性	含义
name	飞机名称。如“玩家 1”或“敌机 1”
max_speed	飞机的最大移动速度
position	飞机的位置。通常是一个包含 x 和 y 坐标的字典，表示飞机在游戏世界中的位置
bullet_range	飞机发射子弹的最大射程
total_planes	类属性，记录游戏中飞机的总数。每次创建新飞机时，这个值都会增加

表 3.10　Plane 类的方法及其含义

方法	含义
__init__()	初始化飞机实例。接收飞机的名称、最大速度、初始位置和子弹射程
move()	根据方向和速度移动飞机，如 up、down、left、right
fire()	飞机向指定方向发射子弹。返回一个子弹对象

（2）Enemy 类（敌机类）：敌方飞机，是 Plane 类的子类，具有额外的得分值属性。Enemy 类的属性及其含义如表 3.11 所示，Enemy 类的方法及其含义如表 3.12 所示。

表 3.11　Enemy 类的属性及其含义

属性	含义
score_value	敌机被摧毁后的得分值。当玩家摧毁敌机时，会根据这个值增加玩家的得分

表 3.12　Enemy 类的方法及其含义

方法	含义
__init__()	初始化敌机实例。除接收父类的构造参数以外，还接收分值
move()	敌机随机移动。在游戏中，敌机会不断改变位置，增加游戏的难度
__str__()	敌机信息的字符串所示

（3）Player 类（玩家飞机类）：玩家控制的飞机，是 Plane 类的子类，具有得分和生命值属性，以及处理生命值和游戏结束的私有方法。Player 类的属性及其含义如表 3.13 所示，Player 类的方法及其含义如表 3.14 所示。

表 3.13　Player 类的属性及其含义

属性	含义
score	玩家的得分。玩家摧毁敌机时，根据敌机的得分值增加自己的得分
__lives	私有属性，玩家的生命值。表示玩家还剩下多少生命值

表 3.14　Player 类的方法及其含义

方法	含义
__init__()	初始化玩家飞机实例。接收玩家的名称、生命值、速度、初始位置和初始得分
__lose_life()	私有方法，玩家失去一条生命。当玩家被击中时，调用这个方法减少生命值
__game_over()	私有方法，游戏结束。当玩家的生命值耗尽时，调用这个方法
__str__()	玩家信息的字符串表示

（4）Bullet 类（子弹类）：描述子弹的移动和伤害，以及限制子弹速度的最大速度类属性。Bullet 类的属性及其含义如表 3.15 所示，Bullet 类的方法及其含义如表 3.16 所示。

表 3.15　Bullet 类的属性及其含义

属性	含义
plane	发射该子弹的飞机对象
position	子弹的位置。通常是一个包含 x 和 y 坐标的字典
direction	子弹发射的方向，如 up、down、left、right
brange	子弹的射程
max_brange	类属性，所有子弹的最大射程。确保子弹射程不超过这个值

表 3.16　Bullet 类的方法及其含义

方法	含义
__init__()	初始化子弹实例。接收子弹的位置、方向和射程
__str__()	子弹信息的字符串表示

（5）GameControl 类（游戏控制类）：管理游戏的主要逻辑，如添加飞机和子弹，以及控制游戏的开始和结束。GameControl 类的属性及其含义如表 3.17 所示，GameControl 类的方法及其含义如表 3.18 所示。

表 3.17　GameControl 类的属性及其含义

属性	含义
players	列表，用于存储游戏中所有的玩家飞机对象。这允许游戏控制器跟踪游戏中的所有活动玩家飞机，并且在需要时进行更新或删除
enemies	列表，用于存储游戏中所有的敌机对象。这允许游戏控制器跟踪游戏中的所有活动敌机，并且在需要时进行更新或删除
bullets	列表，用于存储游戏中发射的所有子弹对象。这样可以管理所有在游戏中移动的子弹，并且用于计算分数

表 3.18　GameControl 类的方法及其含义

方法	含义
__init__()	创建一个新的游戏控制实例。初始化 players、enemies 和 bullets 列表
add_plane()	将一个新的飞机对象添加到 players 或 enemies 列表中。该方法允许游戏动态地添加飞机
add_bullet()	将一个新的子弹对象添加到 bullets 列表中。每当飞机发射子弹时，这个方法都会被调用
update()	更新当前所有飞机和子弹的状态。每一回合结束时都应该调用该方法。该方法判断是否有玩家击中敌机，并且计算分数
start_game()	开始游戏的方法。可以在这里初始化游戏开始时的状态，如重置得分和玩家生命值
end_game()	结束游戏的方法。当游戏结束条件被触发时（如玩家生命值耗尽），这个方法会被调用
show_status()	显示当前所有飞机和子弹的状态信息

程序持续运行直到成功摧毁敌机的最后几行输出结果如图 3.6 所示。

```
玩家移动...
当前玩家状态:
玩家: 玩家1, 位置: {'x': 40, 'y': 80}, 得分: 0, 生命值: 3
当前敌机状态:
敌机: 敌机1, 位置: {'x': 50, 'y': 80}, 分值: 10
当前子弹状态:
敌机移动...
当前玩家状态:
玩家: 玩家1, 位置: {'x': 40, 'y': 80}, 得分: 0, 生命值: 3
当前敌机状态:
敌机: 敌机1, 位置: {'x': 40, 'y': 80}, 分值: 10
当前子弹状态:
玩家射击...
当前玩家状态:
玩家: 玩家1, 位置: {'x': 40, 'y': 80}, 得分: 0, 生命值: 3
当前敌机状态:
敌机: 敌机1, 位置: {'x': 40, 'y': 80}, 分值: 10
当前子弹状态:
玩家1的子弹, 位置: {'x': 40, 'y': 80}, 方向: right, 射程: 20
击中敌机!
游戏结束! 玩家得分:
玩家1得分: 10
```

图 3.6　程序持续运行直到成功摧毁敌机的最后几行输出结果

项目考核

【选择题】

1．面向对象编程（OOP）的主要特点是什么？（　　）。

A．函数　　B．模块化

C．封装、继承和多态　　D．顺序执行

2．在 Python 中定义一个类的语法是什么？（　　）

A．class ClassName:　　B．def ClassName():

C．ClassName = class{}　　D．new ClassName()

3．如何在 Python 中创建一个类的实例？（　　）

A．ClassName()　　B．new ClassName

C．class ClassName:　　D．ClassName.new()

4．下面哪个是 Python 中的构造函数？（　　）

A. __init__()　　B. __construct__()

C. __new__()　　D. __start__()

5．在 Python 中，如何定义一个私有属性？（　　）

A．使用单下划线，如_variable

B．使用双下划线，如__variable

C．使用关键字 private，如 private variable

D．使用关键字 var，如 var variable

6．在 Python 中，如何定义一个类的方法？（　　）

A．def method_name:　　B．method method_name:

C．def method_name():　　D．method_name = def():

7．在 Python 中，如何继承一个类？（　　）

A．class ChildClass(BaseClass):　　B．class BaseClass(ChildClass):

C．class ChildClass extends BaseClass:　　D．class BaseClass extends ChildClass:

8．在 Python 中，如何调用父类的方法？（　　）

A．super().method_name()　　B．parent.method_name()

C．base.method_name()　　D．this.method_name()