面向对象程序设计AI使用建议

Day1：选题+自主学习，提交《游戏选型说明》

一、核心提问方向与具体问题（按Day1任务环节分类）

（一）选题决策类：解决“选哪款游戏”的核心疑问

（二）自主学习类：为后续编码打基础，解决“学什么、怎么学”

（三）选型说明撰写类：解决“怎么写才符合要求”的问题

二、提问的核心原则（确保AI回答精准、可落地）

1. 绑定“场景约束”，避免模糊提问

2. 聚焦“具体需求”，明确输出形式

3. 遵循“认知逻辑”，从“是什么→怎么做”递进

4. 关联“任务要求”，呼应基础/进阶要求

三、总结：Day1提问的核心目标

Day2：游戏设计，完成设计方案初稿

一、核心提问方向与具体问题（按Day2设计方案核心模块分类）

二、提问的核心原则（确保设计方案“可落地、符要求”）

1. 绑定“具体游戏+核心场景”，避免泛化设计

2. 明确“输出形式+报告适配性”，降低方案撰写成本

3. 关联“任务书要求”，提前对齐基础/进阶指标

4. 聚焦“问题解决+设计依据”，支撑报告阐述

三、总结：Day2提问的核心目标

Day3：编码实现与测试

一、核心提问方向与具体问题（按Day3任务核心环节分类）

二、提问的核心原则（确保AI回答“可落地、符要求、易追溯”）

1. 绑定“三重约束”，避免回答空泛

2. 明确“输出形式”，降低落地成本

3. 关联“任务书要求”，提前对齐评分标准

4. 聚焦“问题解决+逻辑说明”，支撑报告与答辩

5. 预留“AI辅助痕迹标注”空间

三、总结：Day3提问的核心目标

Day4：优化代码，撰写课程设计报告
一、核心提问方向与具体问题（按Day4任务核心环节分类）
二、提问的核心原则（确保AI回答“可落地、符规范、防抄袭”）
1. 绑定“前期成果+具体需求”，避免空泛
2. 明确“输出形式+报告适配性”，降低撰写成本
3. 关联“任务书要求”，对齐评分标准
4. 强调“个人化+去AI化”，避免抄袭风险
5. 聚焦“问题解决+逻辑支撑”，便于答辩
三、总结：Day4提问的核心目标

Day1：选题+自主学习，提交《游戏选型说明》

一、核心提问方向与具体问题（按Day1任务环节分类）

（一）选题决策类：解决“选哪款游戏”的核心疑问

这类问题聚焦“游戏适配性”，帮助用户快速锁定符合“规则清晰、面向对象友好、3天可实现”要求的游戏，避免选题过大或适配性差。

具体提问示例	涉及的核心知识方面	提问技巧（为何这样问更优）
“在坦克大战、贪吃蛇、飞机大战、推箱子中，哪款更适合面向对象课程设计？请从‘核心实体数量、封装/继承/多态适配性、编码工作量、3天完成可行性’四个维度对比分析，我熟悉Python语言”	1. 游戏设计复杂度评估；2. 面向对象三大特性（封装、继承、多态）的场景适配；3. 编程语言与游戏开发库的匹配度；4. 短期项目工作量估算	1. 限定候选游戏（任务书推荐范围），避免AI推荐无关游戏；2. 明确评估维度（贴合课程设计核心要求）；3. 补充个人基础（熟悉Python），让推荐更精准；4. 绑定“3天完成”的时间约束，避免选题过大
“贪吃蛇的核心实体有哪些？能否抽象为独立类体现封装和继承？后续新增‘特殊食物’‘障碍墙’功能是否容易扩展？”	1. 游戏实体抽象逻辑；2. 封装（属性私有化、行为封装）的应用场景；3. 继承（父类–子类扩展）的适配性；4. 代码扩展性设计（进阶要求）	1. 聚焦“面向对象特性落地”，避免泛泛而谈；2. 关联进阶要求的“扩展性”，提前规避后续修改难题；3. 针对具体游戏提问，AI回答更具体（而非通用理论）

“选推箱子做面向对象课程设计，可能遇到哪些技术难点？核心实体和交互逻辑是否足够支撑封装、继承特性的体现？”	1. 特定游戏的技术痛点（如路径判断、碰撞检测）；2. 面向对象特性的场景落地可行性；3. 游戏规则复杂度与编码难度的平衡	1. 针对单个候选游戏做“风险预判”，帮助用户规避坑；2. 紧扣课程设计核心要求（体现面向对象特性），避免选“实体少、难抽象”的游戏

（二）自主学习类：为后续编码打基础，解决“学什么、怎么学”

这类问题聚焦“技术栈适配”和“核心知识点快速掌握”，确保用户Day1的自主学习有明确目标，不盲目。

具体提问示例	涉及的核心知识方面	提问技巧（为何这样问更优）
“用Python开发贪吃蛇，推荐什么游戏开发库？请列出‘1天内可掌握的核心知识点’（如窗口创建、蛇移动、碰撞检测），并提供对应的快速入门教程或步骤拆解”	1. 编程语言（Python）与游戏开发库（Pygame）的适配；2. 游戏开发核心技术（窗口渲染、事件监听、坐标计算、碰撞检测）；3. 短期高效学习路径设计	1. 绑定“Python+具体游戏”，技术栈推荐不泛化；2. 限定“1天内掌握”，避免AI提供冗长教程，聚焦核心；3. 明确需要“教程/步骤拆解”，确保学习可落地
“Java开发飞机大战，如何用Swing创建游戏窗口？键盘持续输入（控制飞机移动）和子弹发射的核心代码逻辑是什么？请给出简洁示例（无需完整项目）”	1. Java Swing库的GUI开发基础；2. 事件监听机制（键盘输入处理）；3. 游戏角色运动逻辑（坐标更新、帧刷新）；4. 简单代码片段的解读与应用	1. 聚焦“具体技术点+代码示例”，避免理论说教；2. 明确“无需完整项目”，符合Day1“自主学习预热”的定位；3. 针对“核心玩法”（移动、开火）提问，贴合基础要求
“面向对象游戏开发中，‘碰撞检测’的通用逻辑是什么？用Python/Pygame实现贪吃蛇（蛇撞边界、蛇撞自身）的碰撞检测，需要哪些步骤和关键代码？”	1. 碰撞检测的数学逻辑（坐标对比、矩形交集）；2. 面向对象思想下的碰撞检测封装（独立方法/类）；3. 特定场景（边界碰撞、自身碰撞）的实现差异	1. 先问“通用逻辑”再问“具体实现”，符合“自主学习”的认知规律；2. 绑定具体游戏场景，避免抽象；3. 明确“步骤+关键代码”，帮助用户快速理解核心

（三）选型说明撰写类：解决“怎么写才符合要求”的问题

这类问题聚焦“结构化表达”和“内容填充”，确保《游戏选型说明》包含任务书要求的所有核心要素，且逻辑清晰。

具体提问示例	涉及的核心知识方面	提问技巧（为何这样问更优）
“请提供一份《游戏选型说明》模板，需包含‘选择原因、核心玩法、核心实体、交互逻辑’四个部分，适配贪吃蛇游戏，语言正式符合课程设计报告规范”	1. 课程设计报告的结构化写作规范；2. 游戏核心要素（玩法、实体、交互）的文字表达；3. 正式书面语的运用	1. 明确模板的核心模块（贴合任务书要求），避免AI遗漏内容；2. 绑定具体游戏（贪吃蛇），模板可直接填充修改，降低用户工作量；3. 强调“报告规范”，避免口语化表述
“以坦克大战为例，撰写《游戏选型说明》中的‘核心实体’和‘交互逻辑’部分，核心实体需体现面向对象的封装思想（说明属性和行为），交互逻辑需清晰描述玩家、敌人、子弹、障碍物的互动关系”	1. 游戏实体的抽象能力（属性→状态、行为→功能）；2. 封装思想的文字表达（属性私有化、行为封装为方法）；3. 多实体交互的逻辑梳理（输入→响应→结果）	1. 要求“体现封装思想”，贴合基础要求；2. 明确“多实体交互”，帮助用户梳理复杂逻辑；3. 以具体游戏为例，回答可直接参考，避免用户无从下手
“《游戏选型说明》的‘选择原因’部分，如何结合面向对象课程设计的要求撰写？请以‘选择飞机大战’为例，给出3条具体理由（需包含实体抽象、特性体现、编码可行性）”	1. 课程设计要求与游戏选型的关联性分析；2. 书面化理由的逻辑组织；3. 个人适配性与项目要求的结合	1. 紧扣“课程设计要求”，避免理由泛化（如“我喜欢这款游戏”）；2. 明确理由维度（实体抽象、特性体现、编码可行性），符合学术规范；3. 提供具体示例，帮助用户模仿撰写

二、提问的核心原则（确保AI回答精准、可落地）

1. 绑定“场景约束”，避免模糊提问

- 必带约束条件：课程设计属性（面向对象、3天编码周期、可运行核心玩法）、个人基础（如“熟悉Python”“Java入门”）、游戏范围（任务书推荐列表）。
- 反例：“选什么游戏好？”（无约束，AI可能推荐复杂游戏）；
- 正例：“Java入门水平，3天完成面向对象课程设计，推箱子和贪吃蛇选哪个？请从实体抽象和编码难度分析”。

2. 聚焦“具体需求”，明确输出形式

- 明确要求AI提供“对比表”“模板”“步骤拆解”“代码片段”等具体输出，避免AI给出冗长理论。

- 反例：“告诉我贪吃蛇的面向对象设计”（输出形式模糊）；
- 正例：“请用表格对比贪吃蛇的3个核心实体（蛇、食物、边界）的属性和行为，体现封装思想”。

3. 遵循“认知逻辑”，从“是什么→怎么做”递进

- 先问“选型对比”（是什么游戏适合），再问“技术栈/知识点”（怎么学），最后问“选型说明撰写”（怎么写），符合Day1的任务流程。
- 反例：先问“怎么写选型说明”，再问“选什么游戏”（逻辑倒置，回答无法落地）。

4. 关联“任务要求”，呼应基础/进阶要求

- 提问时主动关联任务书要求（如“体现封装/继承”“代码可扩展”“异常处理基础”），让AI的回答提前适配后续环节。
- 示例：“选坦克大战时，核心实体如何设计才能方便后续添加‘道具系统’（符合进阶要求的扩展性）？”

三、总结：Day1提问的核心目标

所有问题都围绕“快速锁定合适游戏、掌握入门技术、完成规范选型说明”展开，涉及的知识覆盖**面向对象核心特性、游戏开发基础（语言+库+技术点）、报告结构化写作**三大维度。通过“场景约束+具体需求+逻辑递进”的提问方式，可让AI的回答直接服务于Day1的交付物（《游戏选型说明》），同时为Day2的类设计、Day3的编码铺垫基础。

Day2：游戏设计，完成设计方案初稿

一、核心提问方向与具体问题（按Day2设计方案核心模块分类）

Day2的核心是产出“设计方案初稿”，需覆盖**类图设计、面向对象特性落地、设计模式选型、异常处理与扩展性设计**四大核心模块。以下问题聚焦“设计落地性”，避免空泛理论，直接服务于方案撰写。

提问方向	具体提问示例	涉及的核心知识方面	提问技巧（为何这样问更优）

（一）类图设计：解决“核心类怎么抽象、类间关系怎么定”	“以坦克大战为例，设计符合面向对象思想的UML类图，需包含玩家坦克、敌人坦克、子弹、障碍物（墙/河）、道具（血包/强化弹）、游戏管理器6个核心类。请标注每个类的属性（需体现封装，如私有属性）、行为（方法），并说明类间关系（继承/组合/聚合）及设计依据（如“敌人坦克继承自坦克类，因共享移动/开火行为”）”	1. 游戏实体抽象逻辑（属性→状态、行为→功能）；2. UML类图规范（类、属性、方法、关系标注）；3. 继承/组合/聚合的区别与适用场景；4. 封装原则（属性私有化、行为封装为方法）	1. 明确核心类清单（贴合游戏场景），避免AI遗漏关键实体；2. 强制要求“封装体现”“关系说明+设计依据”，贴合基础要求；3. 绑定具体游戏，类图设计可直接落地，无需用户二次拆解
	“贪吃蛇游戏中，‘蛇身’是作为Snake类的内部属性（组合关系），还是独立为SnakeBody类（聚合关系）？请从代码扩展性（如后续新增‘蛇身变色’功能）和逻辑合理性分析，给出最终类设计（含属性和方法）”	1. 组合与聚合的核心区别（生命周期依赖关系）；2. 类设计的逻辑合理性与扩展性平衡；3. 面向对象的模块化思想	1. 聚焦“类间关系争议点”，帮助用户解决实际设计困惑；2. 关联进阶要求的“扩展性”，提前规避后续修改；3. 要求“分析+最终设计”，既懂原理又得结果

	“请用文本描述飞机大战的核心类图（无需画图，用文字说明类名、属性、方法、类间关系），要求体现‘飞机’与‘子弹’的组合关系（飞机发射子弹）、‘特殊子弹’与‘普通子弹’的继承关系，且所有类的属性均为私有（体现封装）”	1. 文本化类图的规范表达（适配报告撰写）；2. 封装原则的落地（私有属性+公开访问器）；3. 组合关系（整体与部分生命周期绑定）的应用场景	1. 要求“文本描述”，直接可复制到设计方案中，降低用户工作量；2. 明确指定关键类间关系，避免AI设计偏离面向对象核心；3. 聚焦“封装+关系”，贴合基础要求
（二）面向对象特性落地：解决“如何在设计中体现封装、继承、多态”	“以推箱子为例，说明你的类设计中如何体现面向对象的三大特性：1. 封装（具体哪个类、哪个属性/行为被封装）；2. 继承（如“特殊箱子继承自箱子类”，需说明共享行为和扩展行为）；3. 多态（如“不同类型的墙对箱子的阻挡效果不同”，如何通过多态实现）”	1. 封装、继承、多态的核心定义与游戏场景适配；2. 多态的实现逻辑（方法重写、父类引用指向子类对象）；3. 特性落地的“场景化表达”（避免纯理论）	1. 要求“分点说明+具体场景”，避免AI泛泛而谈；2. 绑定具体游戏，特性体现更具象（如推箱子的“多态”对应不同墙的阻挡逻辑）；3. 直接呼应报告要求“说明面向对象特性”，可直接作为方案内容
	“贪吃蛇的‘普通食物’和‘特殊食物’（如加速食物、变长食物），如何通过继承设计体现代码复用？请给出Food父类和子类的属性、方法，说明继承的设计价值（解决了什么问题）”	1. 继承的核心价值（代码复用、统一接口）；2. 父类与子类的职责划分（父类封装共性，子类扩展特性）；3. 面向对象的“开闭原则”（对扩展开放，对修改关闭）	1. 聚焦“继承的实际应用场景”，避免用户只懂理论不会用；2. 要求“设计+价值说明”，既完成设计又能在报告中阐述“为什么用”；3. 关联进阶要求的扩展性，为后续新增食物类型铺垫

（三）设计模式选型：解决“选什么模式、怎么用、为什么用”	“坦克大战中，需支持‘普通子弹、穿甲弹、爆炸弹’三种子弹类型，且后续可能新增激光弹。请推荐1种设计模式（如工厂模式），说明：1. 为什么选该模式（解决什么问题，如“避免新增子弹时修改发射逻辑”）；2. 模式的核心角色与游戏类的对应关系（如“子弹工厂对应BulletFactory类”）；3. 关键设计代码片段（无需完整实现，体现模式核心逻辑）”	1. 工厂模式（简单工厂/工厂方法）的适用场景（对象创建复杂、需统一管理）；2. 设计模式与游戏场景的适配性；3. 模式核心角色的落地（工厂类、产品类、客户端）；4. 开闭原则的应用（扩展新增类，不修改原有代码）	1. 明确“场景+扩展需求”，让设计模式选型有依据；2. 要求“原因+角色对应+代码片段”，覆盖报告中“为什么用”“怎么用”的阐述需求；3. 绑定具体子弹场景，模式落地更清晰，避免抽象
	“飞机大战的‘碰撞检测’场景（飞机撞敌人、飞机撞道具、子弹撞敌人），每种碰撞的处理逻辑不同（如撞敌人游戏结束，撞道具加分）。请推荐1种设计模式（如策略模式），说明模式如何解决“碰撞逻辑多样化”的问题，给出模式与游戏类的对应关系及核心设计思路”	1. 策略模式的适用场景（算法/逻辑多样化，需灵活切换）；2. 设计模式解决的核心痛点（代码冗余、耦合度高）；3. 模式与面向对象特性的结合（如策略接口+具体策略类，体现多态）	1. 聚焦“具体业务场景”（碰撞检测），避免模式选型泛化；2. 要求“解决问题+设计思路”，直接支撑报告中“设计模式应用”部分；3. 关联进阶要求的“扩展性”，符合任务书要求

	“贪吃蛇游戏中，需实时监听‘蛇撞边界、蛇撞自身、蛇吃食物’三种事件，触发不同响应（游戏结束、蛇增长）。请推荐1种设计模式（如观察者模式），说明模式如何降低事件触发与响应的耦合度，给出模式核心类的设计（如“主题类对应Snake类，观察者类对应GameOverListener、SnakeGrowListener”）”	1. 观察者模式的适用场景（事件通知、一对多依赖）；2. 解耦的核心思想（降低类间直接依赖）；3. 游戏事件驱动模型的设计逻辑	1. 明确“事件场景+痛点（耦合）”，让模式选型有针对性；2. 要求“类设计对应”，避免模式只停留在理论；3. 贴合进阶要求的“代码扩展性”，后续新增事件（如蛇撞障碍）无需修改原有逻辑
（四）异常处理与扩展性设计：适配进阶要求	“坦克大战中，常见异常场景有‘坦克撞边界、非法键盘输入（如同时按上下键）、道具加载失败、子弹超出屏幕未销毁’。请设计异常处理逻辑，说明：1. 每个异常场景的处理方式（如“撞边界则禁止移动”）；2. 如何通过类设计封装异常处理（如是否单独设计CollisionException类，或在核心类中内置checkException方法）”	1. 游戏常见异常场景识别；2. 异常处理的设计原则（预判异常、友好处理、不崩溃）；3. 异常处理的封装逻辑（避免散落在主函数）；4. 面向对象的模块化思想	1. 明确列出核心异常场景，避免AI遗漏；2. 要求“处理方式+类设计”，既解决问题又符合面向对象规范；3. 直接呼应进阶要求的“异常处理”，无需用户额外思考场景

	“贪吃蛇游戏要满足‘新增特殊道具（如减速道具）时，无需修改Snake类和Game类’的扩展性要求。请设计类结构（含核心类的属性和方法），说明如何通过面向对象原则（如开闭原则）和设计模式实现该扩展能力”	1. 开闭原则的落地逻辑（扩展新增类，不修改原有类）；2. 模块化设计（功能拆分到独立类）；3. 设计模式与扩展性的结合（如策略模式/工厂模式）	1. 明确“扩展场景+约束条件”，让设计有明确目标；2. 要求“类结构+原则/模式说明”，既完成设计又能在报告中阐述“扩展性设计依据”；3. 紧扣进阶要求，提前规避后续修改难题

二、提问的核心原则（确保设计方案“可落地、符要求”）

1. 绑定“具体游戏+核心场景”，避免泛化设计

- 所有提问必须明确具体游戏（如坦克大战、贪吃蛇），且聚焦游戏核心场景（如子弹发射、碰撞检测、道具扩展），避免AI给出“通用类图”“通用模式”，导致无法直接应用。
- 反例：“如何设计游戏的类图？”（无约束，AI回答空泛）；
- 正例：“推箱子游戏中，箱子、玩家、目标点的类设计如何体现封装？请给出属性、方法及类间关系”。

2. 明确“输出形式+报告适配性”，降低方案撰写成本

- 要求AI提供“文本类图”“表格对比设计模式”“分点说明设计依据”等结构化输出，可直接复制到设计方案中，无需用户二次整理。
- 示例：“请用表格列出你推荐的2种设计模式（适配飞机大战），包含‘模式名称、核心角色、游戏类对应、解决的问题’四列”。

3. 关联“任务书要求”，提前对齐基础/进阶指标

- 提问时主动呼应任务书要求，如“体现封装”“符合开闭原则”“适配扩展性”“异常处理”，确保AI的设计不偏离评分标准。
- 示例：“设计贪吃蛇的类时，需体现封装（私有属性+公开方法）和继承（特殊食物继承自食物类），并说明如何支持后续新增‘有毒食物’（扩展性要求）”。

4. 聚焦“问题解决+设计依据”，支撑报告阐述

- 不仅要求AI给出“设计结果”（如类图、模式选型），还需说明“为什么这么设计”“解决了什么问题”，直接支撑报告中“设计依据”“模式应用价值”的撰写。
- 反例：“推荐坦克大战的设计模式”（仅要结果，无依据）；
- 正例：“坦克大战的道具系统推荐什么设计模式？说明该模式解决了什么问题（如“避免新增道具时修改道具生成逻辑”），以及模式核心角色与游戏类的对应关系”。

三、总结：Day2提问的核心目标

所有问题都围绕“设计方案初稿的核心要素”展开，涉及的知识覆盖**UML类图规范、面向对象三大特性落地、设计模式场景适配、异常处理逻辑、开闭原则与扩展性设计**五大维度。通过“具体场景+结构化输出+任务书对齐”的提问方式，可让AI的回答直接转化为设计方案的核心内容，同时帮助用户理解设计背后的逻辑（便于后续答辩）。

Day3：编码实现与测试

一、核心提问方向与具体问题（按Day3任务核心环节分类）

Day3的核心目标是**将设计方案落地为可运行代码+完成核心功能/异常场景测试**，提问需聚焦“编码落地、设计模式实现、异常处理、测试验证、问题排查”五大核心环节，所有问题均绑定“具体游戏+技术栈+设计方案”，确保AI回答可直接落地，同时贴合任务书“类设计、设计模式、异常处理、代码规范”要求。

提问方向	具体提问示例	涉及的核心知识方面	提问技巧（为何这样问更优）

（一）核心类编码实现：将设计方案的类图转化为可运行代码	“我用Python+Pygame开发贪吃蛇，Day2设计的核心类包括：Snake（属性：私有蛇身坐标列表、移动方向；行为：move()、grow()、checkCollision()）、Food（属性：私有坐标、类型；行为：generate()）。请给出这两个类的完整编码，要求：1. 体现封装（属性私有化，通过getter/setter访问）；2. move()方法实现蛇自动前进（基于方向更新坐标）；3. 关键代码添加注释，符合Python编码规范”	1. 面向对象封装的代码落地（私有属性、访问器方法）；2. Python语法（类定义、私有变量、方法参数设计）；3. Pygame核心API使用（坐标计算、帧刷新逻辑）；4. 游戏核心行为逻辑（移动、食物生成、碰撞检测数学原理）；5. 代码规范（命名、注释）	1. 绑定“技术栈（Python+Pygame）+已设计类结构”，避免AI生成无关代码；2. 明确“封装+核心行为+代码规范”三大要求，贴合基础要求；3. 要求“完整编码+注释”，可直接复用（需标注AI辅助痕迹）；4. 不要求完整项目，聚焦核心类，符合Day3“编码实现”的阶段性目标

	“Java+Swing开发坦克大战，玩家坦克（PlayerTank）继承自父类Tank，需实现‘移动（上下左右键控制）、开火（空格键发射子弹）’功能。请给出PlayerTank类的编码：1. 继承Tank父类（父类含x、y坐标、速度、方向等私有属性，提供get/set方法）；2. 绑定Swing键盘事件监听（处理持续移动和单次开火）；3. 开火时创建Bullet对象（关联Day2设计的子弹类）”	1. 继承特性的代码落地（子类继承父类、重写/扩展方法）；2. Java Swing事件监听机制（键盘输入处理）；3. 类间关联调用（坦克→子弹的对象创建）；4. 游戏角色运动逻辑（坐标更新、按键响应优先级）	1. 衔接Day2的类设计（继承关系），确保编码与设计一致；2. 明确“键盘事件+对象关联”，覆盖核心玩法；3. 给出父类基础信息，避免AI设计冲突；4. 贴合“核心实体抽象为类”的基础要求

（二）设计模式落地编码：将Day2选定的模式转化为代码	“坦克大战中，我用‘简单工厂模式’管理子弹（普通弹、穿甲弹、爆炸弹），Day2设计的核心角色：BulletFactory（工厂类）、BaseBullet（抽象产品类）、CommonBullet/ArmorPiercingBullet（具体产品类）。请用Java实现这4个类的核心代码：1. 工厂类提供createBullet()方法（根据类型创建对应子弹对象）；2. 抽象产品类定义抽象方法fire()，具体子类重写该方法（不同子弹的发射逻辑）；3. 说明如何在PlayerTank的fire()方法中调用工厂类，体现开闭原则（新增激光弹无需修改工厂类和坦克类）”	1. 简单工厂模式的代码实现逻辑（抽象产品、具体产品、工厂类职责划分）；2. 面向对象的抽象与多态（抽象类+方法重写）；3. 开闭原则的落地（扩展新增类，不修改原有代码）；4. 类间依赖关系（工厂类→产品类→客户端类的调用流程）	1. 衔接Day2的设计模式选型，明确核心角色与类名，避免AI偏离设计；2. 要求“核心代码+调用逻辑+原则说明”，既完成编码又支撑报告中“设计模式应用价值”的阐述；3. 绑定“新增子弹”的扩展场景，贴合进阶要求；4. 明确输出形式（类代码+调用说明），直接落地

	“飞机大战的碰撞检测用‘策略模式’，Day2设计：CollisionStrategy（策略接口，含handleCollision()方法）、PlaneEnemyCollision/PlanePropCollision（具体策略类）、CollisionContext（上下文类，持有策略对象）。请用Python实现：1. 策略接口和2个具体策略类（分别实现“飞机撞敌人→游戏结束”“飞机撞道具→加分”逻辑）；2. 上下文类的setStrategy()和executeStrategy()方法；3. 说明如何在Game主循环中调用上下文类，切换不同碰撞策略”	1. 策略模式的代码实现（接口定义、具体策略类、上下文类）；2. 多态特性的落地（接口引用指向子类对象）；3. 游戏主循环的逻辑组织（碰撞检测触发时机）；4. 解耦思想（将不同碰撞逻辑拆分到独立类，避免代码冗余）	1. 明确模式的核心角色与游戏场景，确保编码贴合设计；2. 要求“完整策略链路+调用示例”，覆盖“模式落地+实际使用”；3. 聚焦“解耦”核心痛点，呼应进阶要求的“代码可维护性”；4. 代码片段可直接嵌入项目，减少用户工作量

（三）异常处理编码：落实进阶要求的异常场景处理	“贪吃蛇的常见异常场景：1. 蛇撞边界；2. 同时按下两个相反方向键（非法输入）；3. 食物加载失败（Pygame图片资源不存在）。请用Python实现异常处理逻辑：1. 在Snake的checkCollision()方法中检测边界碰撞，返回布尔值并在Game类中处理（触发游戏结束）；2. 处理非法输入（忽略后按的相反方向键），无需自定义异常类；3. 食物加载时用try-except捕获FileNotFoundError，打印提示并使用默认颜色块替代图片；4. 所有异常处理逻辑封装在对应类的方法中，不散落主函数”	1. 异常处理机制（try-except捕获、自定义异常可选）；2. 游戏异常场景的预判与处理原则（友好提示、不崩溃、逻辑兜底）；3. 面向对象的模块化处理（异常检测/处理封装到对应类）；4. Pygame资源加载的异常类型	1. 明确列出核心异常场景，避免AI遗漏；2. 要求“具体处理方式+代码实现+封装要求”，贴合进阶要求；3. 区分“逻辑异常（边界碰撞）”和“运行时异常（资源加载失败）”，处理方式更精准；4. 代码直接落地，同时符合“按职责划分模块”的基础要求

	“Java开发推箱子，异常场景：1. 箱子撞边界；2. 玩家推动箱子时撞其他箱子（无法移动）；3. 关卡文件读取失败（IO异常）。请设计异常处理代码：1. 自定义CollisionException类（继承Exception），在Box类的move()方法中检测碰撞时抛出；2. 在Player类的pushBox()方法中捕获该异常，提示“无法移动”；3. 读取关卡文件时用try-catch捕获IOException，打印日志并加载默认关卡；4. 说明异常处理的封装逻辑（为何不直接在主函数中捕获）”	1. 自定义异常类的设计（继承关系、构造方法）；2. 异常的抛出与捕获流程（方法抛出、调用者捕获）；3. 面向对象的模块化思想（异常处理与业务逻辑绑定）；4. Java IO异常的类型与处理方式	1. 结合推箱子的核心场景，异常处理更具针对性；2. 要求“自定义异常+捕获处理+逻辑说明”，覆盖进阶要求的核心；3. 问“为何不散落主函数”，帮助用户理解封装的意义，便于报告阐述；4. 代码符合Java编码规范，可直接复用
（四）功能测试与问题排查：验证代码可运行性，解决bug	“我用Python+Pygame实现了贪吃蛇的核心代码（Snake类、Food类、Game主循环），现在测试时遇到两个问题：1. 蛇移动时身体重叠（前一节身体没跟上头部坐标）；2. 吃食物后身体没有增长。请分析可能的原因，给出具体的排查步骤和修改后的代码片段（标注需要修改的部分）”	1. 游戏角色移动的坐标同步逻辑（蛇身节点的迭代更新）；2. 贪吃蛇增长的核心逻辑（添加新节点到蛇身列表）；3. Pygame帧刷新的时序问题；4. 代码调试思路（断点设置、变量打印）	1. 描述具体bug现象，提供足够上下文（技术栈、核心类），AI能精准定位；2. 要求“原因分析+排查步骤+修改代码”，不仅解决问题，还教会用户排查思路；3. 标注修改部分，便于用户追溯和标注AI辅助痕迹；4. 聚焦核心玩法bug，确保游戏可运行

	“坦克大战的核心玩法测试用例应该包含哪些？请按‘核心功能（坦克移动、开火、子弹碰撞障碍物）+ 异常场景（撞边界、非法输入、道具加载失败）’分类，给出每个测试用例的‘测试步骤、预期结果’，适配我用Java+Swing开发的版本（坦克移动速度5px/帧，子弹射程屏幕边界）”	1. 软件测试用例设计（场景覆盖、步骤清晰、预期明确）；2. 游戏核心玩法的逻辑验证点；3. 异常场景的测试覆盖原则；4. 与开发技术栈/参数的适配性	1. 按“核心功能+异常场景”分类，贴合Day3“功能测试”的目标；2. 绑定开发的具体参数（速度、射程），测试用例更精准；3. 要求“步骤+预期结果”，可直接按用例手动测试，无需额外整理；4. 覆盖基础要求（核心玩法可运行）和进阶要求（异常场景处理有效）
	“Pygame开发的飞机大战，运行时出现‘子弹发射后不显示’的问题，核心代码如下（粘贴子弹类和Game主循环的绘图部分）：[此处可模拟粘贴代码]。请帮我定位问题原因，给出修改方案，并说明如何避免后续类似问题（如绘图顺序、对象实例化时机）”	1. Pygame绘图机制（绘图顺序、surface刷新）；2. 游戏对象的实例化与生命周期管理；3. 坐标计算错误排查；4. 代码逻辑漏洞（如子弹列表未添加新实例）	1. 要求用户提供“具体代码片段+bug现象”，AI排查更精准，避免泛泛而谈；2. 要求“原因+修改方案+避坑建议”，既解决当前问题，又预防后续bug；3. 贴合Day3“测试中问题排查”的核心需求，确保游戏可运行；4. 建议部分可写入报告的“设计过程”章节

二、提问的核心原则（确保AI回答“可落地、符要求、易追溯”）

1. 绑定“三重约束”，避免回答空泛

- 必带约束：**具体游戏**（如贪吃蛇、坦克大战）、**技术栈**（如Python+Pygame、Java+Swing）、**已设计方案**（如类名、设计模式选型），确保AI回答与前期设计一致，不偏离项目框架。
- 反例：“如何实现游戏的移动功能？”（无约束，AI回答通用，无法直接复用）；

- 正例：“Python+Pygame开发推箱子，Player类如何实现‘上下左右移动’，要求封装在move()方法中，检测到撞边界则禁止移动，给出代码片段”。

2. 明确“输出形式”，降低落地成本

- 要求AI提供“代码片段（标注关键修改点）、测试用例（步骤+预期）、排查步骤（分点）”等结构化输出，可直接嵌入项目或用于测试，无需用户二次整理。
- 示例：“请给出Snake类checkCollision()方法的代码，关键逻辑（自身碰撞检测）用注释标注，同时说明该代码如何体现封装原则”（输出直接复用，还能支撑报告阐述）。

3. 关联“任务书要求”，提前对齐评分标准

- 提问时主动呼应基础/进阶要求，如“体现封装”“符合开闭原则”“异常处理封装到类”“代码规范带注释”，确保AI回答不偏离核心评分点。
- 示例：“实现子弹工厂类时，需符合开闭原则（新增子弹无需修改工厂类），代码命名符合Java规范，关键步骤添加注释”。

4. 聚焦“问题解决+逻辑说明”，支撑报告与答辩

- 不仅要求AI给出“结果（代码/用例）”，还需说明“原因（为何这样写）”“解决的问题（如‘避免代码冗余’）”，直接支撑报告中“设计过程”“总结”章节的撰写，同时帮助用户理解逻辑（便于答辩）。
- 反例：“给出坦克开火的代码”（仅要结果，无逻辑）；
- 正例：“给出PlayerTank的fire()方法代码，调用BulletFactory创建子弹，说明这样设计如何解决‘新增子弹时修改坦克类’的问题”。

5. 预留“AI辅助痕迹标注”空间

- 提问时可要求AI在代码中标注“可标注AI辅助的部分”，或在回答中说明“该代码可标注为‘参考AI生成的工厂类逻辑，未修改核心结构’”，贴合交付物“标注AI辅助痕迹”的要求。
- 示例：“请给出CollisionStrategy接口的Python代码，在代码开头添加注释‘// AI辅助生成：策略模式核心接口定义，未修改’，便于后续标注”。

三、总结：Day3提问的核心目标

所有问题均围绕“**代码可运行、功能可测试、符合设计要求**”展开，涉及的知识覆盖**面向对象编码落地、设计模式实现、异常处理机制、游戏开发库使用、软件测试用例设计、bug排查思路**六大维度。通过“三重约束+结构化输出+任务书对齐”的提问方式，既能让AI高效解决编码、测试中的实际问题，又能为后续Day4的报告撰写和Day5的答辩铺垫基础，同时满足“标注AI辅助痕迹”“代码规范”等交付物要求。

Day4：优化代码，撰写课程设计报告

一、核心提问方向与具体问题（按Day4任务核心环节分类）

Day4的核心目标是**代码优化（贴合进阶要求）+ 完整撰写课程设计报告（覆盖提纲所有要点）**，提问需聚焦“代码重构、性能优化、报告结构化撰写、内容逻辑梳理”四大核心，所有问题均绑定“具体游戏+前期设计/编码成果”，确保AI回答可直接落地，同时符合任务书“代码规范、扩展性、报告规范”要求。

提问方向	具体提问示例	涉及的核心知识方面	提问技巧（为何这样问更优）
（一）代码优化：提升规范度、扩展性、性能（适配进阶要求）	“我用Python+Pygame开发贪吃蛇，已完成核心编码，现在需要优化代码：1. 目前Snake类的checkCollision()方法同时处理‘撞边界’和‘撞自身’，如何按‘单一职责原则’拆分？2. 新增‘减速道具’时，需修改Snake类的move()方法，如何重构才能符合‘开闭原则’（不修改原有代码）？3. 游戏运行时帧速率不稳定（约25–30fps），有哪些Pygame性能优化技巧？请给出具体修改方案和代码片段，标注修改点”	1. 面向对象设计原则（单一职责、开闭原则）的落地；2. 代码重构技巧（方法拆分、模块解耦）；3. Pygame性能优化（帧速率控制、资源复用、绘制逻辑简化）；4. 代码扩展性设计（新增功能的模块化实现）；5. 代码规范（方法职责划分、命名优化）	1. 绑定“已完成编码+具体问题”，避免AI给出泛化优化建议；2. 明确“设计原则+性能”两大优化方向，贴合进阶要求；3. 要求“修改方案+代码片段+标注”，可直接落地，同时便于标注AI辅助痕迹；4. 聚焦实际痛点（帧速率、扩展性），优化有明确目标

	“Java+Swing开发坦克大战，代码存在以下问题：1. 子弹发射逻辑散落在PlayerTank类中，与BulletFactory耦合过高；2. 异常处理（如道具加载失败）的提示信息不规范；3. 类名存在拼音（如DanZi类应为Bullet）。请给出优化方案：1. 解耦坦克与子弹工厂的依赖；2. 统一异常提示格式（含错误类型、发生位置）；3. 修正命名并补充缺失的注释。要求符合Java编码规范（如驼峰命名、注释格式）”	1. 代码解耦技巧（依赖注入、接口隔离）；2. 编码规范（命名规则、注释规范、异常提示规范）；3. Java语法细节（接口使用、依赖管理）；4. 模块化设计（功能逻辑拆分）	1. 列出具体代码问题，AI优化更精准；2. 关联“代码规范”基础要求和“解耦”进阶要求；3. 要求“方案+落地细节”，避免空泛建议；4. 贴合交付物“代码规范”要求，降低答辩风险
	“如何规范标注代码中的‘AI辅助痕迹’？以我参考AI生成的Snake类move()方法为例（原代码：[粘贴AI生成的代码]），我修改了‘蛇身坐标更新逻辑’以符合封装原则，请给出标注示例，要求格式清晰、说明具体（含AI辅助内容、个人修改点、修改原因）”	1. 任务书“AI辅助痕迹标注”要求的解读；2. 代码注释规范（辅助痕迹标注的格式、内容要素）；3. 学术诚信（明确区分AI生成与个人修改）	1. 聚焦任务书强制要求，避免遗漏关键交付物规范；2. 要求“示例+格式”，可直接复用；3. 绑定“具体代码+个人修改”，标注更真实可落地

（二）报告撰写：结构化填充+逻辑梳理（覆盖报告提纲所有要点）	1. 设计简介模块	“请以‘Python+Pygame贪吃蛇’为例，撰写报告的‘1.1 游戏选型说明’，包含：1. 选择原因（结合面向对象适配性、编码可行性）；2. 核心玩法（分点描述，含操作方式、胜负规则）；3. 核心实体（蛇、食物、道具）及属性/行为；4. 交互逻辑（用文字描述‘玩家输入→蛇移动→碰撞检测→结果响应’的链路）。要求语言正式、逻辑清晰，符合课程设计报告规范”	1. 报告结构化写作（按提纲要求组织内容）；2. 游戏核心要素的文字表达（实体抽象、交互逻辑梳理）；3. 正式书面语规范（避免口语化）；4. 面向对象思想与游戏选型的关联性阐述

	2. 设计方案模块	“我用坦克大战的设计方案撰写‘2. 设计方案’章节：1. 已设计核心类（Tank、PlayerTank、EnemyTank、Bullet、BulletFactory、CollisionStrategy），请用文字描述类图（含类间关系：继承、组合、实现，如PlayerTank继承Tank，Tank组合BulletFactory）；2. 说明代码中体现的封装、继承、多态（各举1个具体类/方法示例）；3. 应用了‘简单工厂模式’（子弹创建）和‘策略模式’（碰撞检测），请分别说明‘为什么用’（解决的问题）和‘怎么用’（核心角色对应关系）。要求语言学术化，避免口语化”	1. UML类图的文字化表达（类、属性、方法、类间关系描述）；2. 面向对象特性的场景化阐述（结合具体代码）；3. 设计模式的应用价值分析（问题–方案–效果）；4. 学术书面语表达（逻辑连贯、专业术语准确）

	3. 设计过程模块	“请帮我梳理报告‘3. 设计过程’的内容框架：1. 按‘设计阶段→编码阶段→测试阶段’划分，每个阶段描述遇到的2个核心问题（如设计阶段‘类间关系混淆’、编码阶段‘键盘事件监听冲突’、测试阶段‘子弹穿透障碍物’）；2. 记录AI工具的使用过程（含3个关键提问、AI的回答、我的甄别/修改思路，如‘提问AI“坦克如何绑定键盘事件”，AI给出的代码未处理持续移动，我添加了keyPressed事件的状态标记’）；3. 要求逻辑连贯，问题与解决办法对应，AI使用记录符合任务书要求”	1. 项目过程的结构化梳理（按阶段划分问题）；2. AI辅助过程的规范记录（提问–回答–甄别–修改）；3. 问题解决逻辑的文字表达（原因–方案–效果）；4. 报告内容的真实性与合理性（问题贴合游戏开发实际）

	4. 设计结果模块	“撰写报告‘4. 设计结果’：1. 以‘坦克大战（Java+Swing）’为例，介绍项目工程文件结构（按‘src/（含entity、factory、strategy、util包）、resource/（图片、音效）、doc/（报告）’分类，说明每个包下的核心类功能）；2. 描述游戏运行效果（含核心功能演示：坦克移动/开火、子弹碰撞、道具生效）和异常场景处理效果（如撞边界、资源加载失败）；3. 要求语言简洁、信息完整，附文件结构示意图（用文本描述层级）”	1. 项目工程结构设计（按功能模块化划分包/文件）；2. 软件运行效果的文字描述（功能完整性、异常处理有效性）；3. 文本化示意图的规范表达（层级清晰、分类合理）

	5. 总结与参考文献模块	“1. 结合贪吃蛇的开发过程，总结面向对象程序设计的3个核心原则（封装、单一职责、开闭原则），要求用自己的语言（禁止复制AI内容），每个原则配1个游戏开发中的具体示例（如封装：Snake类的坐标属性私有化）；2. 总结‘简单工厂模式’和‘策略模式’的核心要点（解决的问题、适用场景）；3. 推荐3篇与‘Python+Pygame游戏开发’‘面向对象设计原则’相关的中英文参考文献，要求符合学术规范（含作者、标题、来源、发表年份），并说明在报告中如何引用”	1. 面向对象设计原则的理解与场景化总结；2. 设计模式的核心逻辑与适用场景提炼；3. 学术参考文献规范（格式、来源可信度）；4. 学术引用规范（正文标注、参考文献列表格式）

（三）报告格式与逻辑优化：确保规范度和连贯性	“我已完成课程设计报告初稿（贪吃蛇），请从以下方面优化：1. 格式：统一标题层级（一级标题黑体三号、二级标题宋体小四加粗）、段落间距（1.5倍）、代码块格式（Python代码用等宽字体、缩进一致）；2. 逻辑：补充‘设计方案’与‘编码实现’的衔接内容（如‘类图中的CollisionStrategy接口，对应编码中的CollisionStrategy.py文件，核心方法为handleCollision()’）；3. 语言：修正口语化表达（如‘然后蛇就会增长’改为‘蛇触发grow()方法实现身体增长’）。请给出具体修改示例和全文格式模板”	1. 课程设计报告格式规范（标题、段落、代码块格式）；2. 报告逻辑连贯性（章节间衔接、内容对应）；3. 专业术语的准确使用（避免口语化）；4. 文档排版技巧（统一格式、视觉清晰）	1. 明确“格式+逻辑+语言”三大优化方向，覆盖报告规范要求；2. 要求“修改示例+模板”，可直接套用；3. 聚焦“衔接内容”，解决报告常见逻辑断层问题；4. 绑定“具体游戏”，修改示例更贴合报告内容

二、提问的核心原则（确保AI回答“可落地、符规范、防抄袭”）

1. 绑定“前期成果+具体需求”，避免空泛

- 必带约束：**具体游戏**（如贪吃蛇、坦克大战）、**技术栈**（如Python+Pygame、Java+Swing）、**前期成果**（已完成编码/报告初稿、已设计的类/模式）、**具体痛点**（如代码耦合、报告逻辑断层），确保AI回答与项目高度匹配。
- 反例：“如何优化游戏代码？”（无约束，AI回答通用）；

- 正例：“Python+Pygame贪吃蛇，Snake类的move()方法与道具逻辑耦合，如何按开闭原则重构？给出代码片段”。

2. 明确“输出形式+报告适配性”，降低撰写成本

- 要求AI提供“结构化输出”（如分点总结、模板、修改示例、文本示意图），可直接复制到报告中，无需二次整理；同时强调“报告规范”（正式书面语、学术格式），避免AI生成口语化或格式混乱的内容。
- 示例：“请用文本层级描述坦克大战的工程结构，格式为‘文件夹名：核心文件/包（功能说明）’，符合报告排版要求”。

3. 关联“任务书要求”，对齐评分标准

- 提问时主动呼应基础要求（代码规范、报告格式、核心玩法）和进阶要求（设计模式、扩展性、异常处理），同时满足报告提纲的所有要点（如AI使用记录、参考文献），确保回答不偏离评分核心。
- 示例：“撰写‘设计过程’时，需包含AI工具的使用记录（提问、回答、修改思路），符合任务书交付物要求”。

4. 强调“个人化+去AI化”，避免抄袭风险

- 针对报告总结、选型原因等章节，明确要求“用自己的语言”“配具体示例”，禁止AI生成通用化、可复制的内容；同时要求AI提供“框架+示例”，用户需补充个人开发细节（如实际遇到的bug、个人修改思路），确保报告真实性。
- 反例：“帮我写报告总结”（易生成通用内容）；
- 正例：“结合我开发贪吃蛇时‘拆分checkCollision()方法’的经历，总结单一职责原则，用自己的语言描述，避免复制AI内容”。

5. 聚焦“问题解决+逻辑支撑”，便于答辩

- 不仅要求AI给出“结果（优化代码、报告内容）”，还需说明“原因（为何这样优化、为何这样总结）”，帮助用户理解背后的逻辑，便于答辩时阐述“设计思路”“优化依据”。
- 示例：“重构Snake类的move()方法时，为何选择新增‘SpeedStrategy接口’？请说明设计依据（如符合开闭原则、便于扩展多种速度道具）”。

三、总结：Day4提问的核心目标

所有问题均围绕“代码优化落地+报告规范完整”展开，涉及的知识覆盖**面向对象设计原则、代码重构、性能优化、报告结构化写作、学术规范、设计模式总结**六大维度。通过“前期成果绑定+结构化输出+任务书对齐+去AI化要求”的提问方式，既能让AI高效解决代码和报告中的实际问题，又能确保最终交付物（优化后的代码、规范的报告）符合任务书要求，同时为Day5的答辩铺垫充足的逻辑支撑。