GameFloaty

概述（结构化提要）

1. 产品定位

> **一句话**：在不离开游戏画面的情况下，向 PC 玩家提供即时攻略、语音交流与上下文感知的 AI 陪玩教练。

- **核心卖点**
  - 零 Alt-Tab 中断：浮窗叠加在游戏上层
  - 多语言 Wiki 整合 + AI 问答：中英双源、自动翻译、精炼答案
  - 语音 & OCR 多模态：说一声即可查，屏幕文字自动识别
  - 可连续对话：类似「随身军师」，支持追问与场景联想

2. 目标用户与痛点

用户	主要痛点	助手价值
硬核探索玩家	复杂机制 + 资料分散	自动抓取当前关卡/物品攻略
竞技进阶玩家	缺开局/Build 优化	实时语音战术建议
轻度休闲玩家	卡关耗时、学习成本高	一句话问答、省时省力
跨语言玩家	资料语言不匹配	自动翻译并呈现母语版本

3. MVP 必备能力

1. **游戏识别**：通过窗口标题/OCR 判定当前游戏
2. **浮窗呈现**：可呼出/隐藏、位置大小可调
3. **快捷触发**：自定义热键呼叫助手
4. **中英 Wiki 检索**：双语信息源打通
5. **AI 问答摘要**：LLM 精炼攻略要点
6. **TTS 语音播报**：边玩边“听”答案
7. **基础上下文追问**：记住上一轮问题，支持继续追问

4. 迭代扩展能力

- OCR+视觉模型识怪/识图 → 无文字界面也能推断情境

- 个性化教练：读取玩家日志/装备，给差异化建议
- 自动笔记 & 任务日志回顾
- 主动式提示：多次失败或进入隐藏区域时智能弹窗
- 第二屏/移动端联动、图片/视频攻略嵌入
- 社区共创：分享配置、语音包、攻略补充

5. 关键交互流程（典型用例）

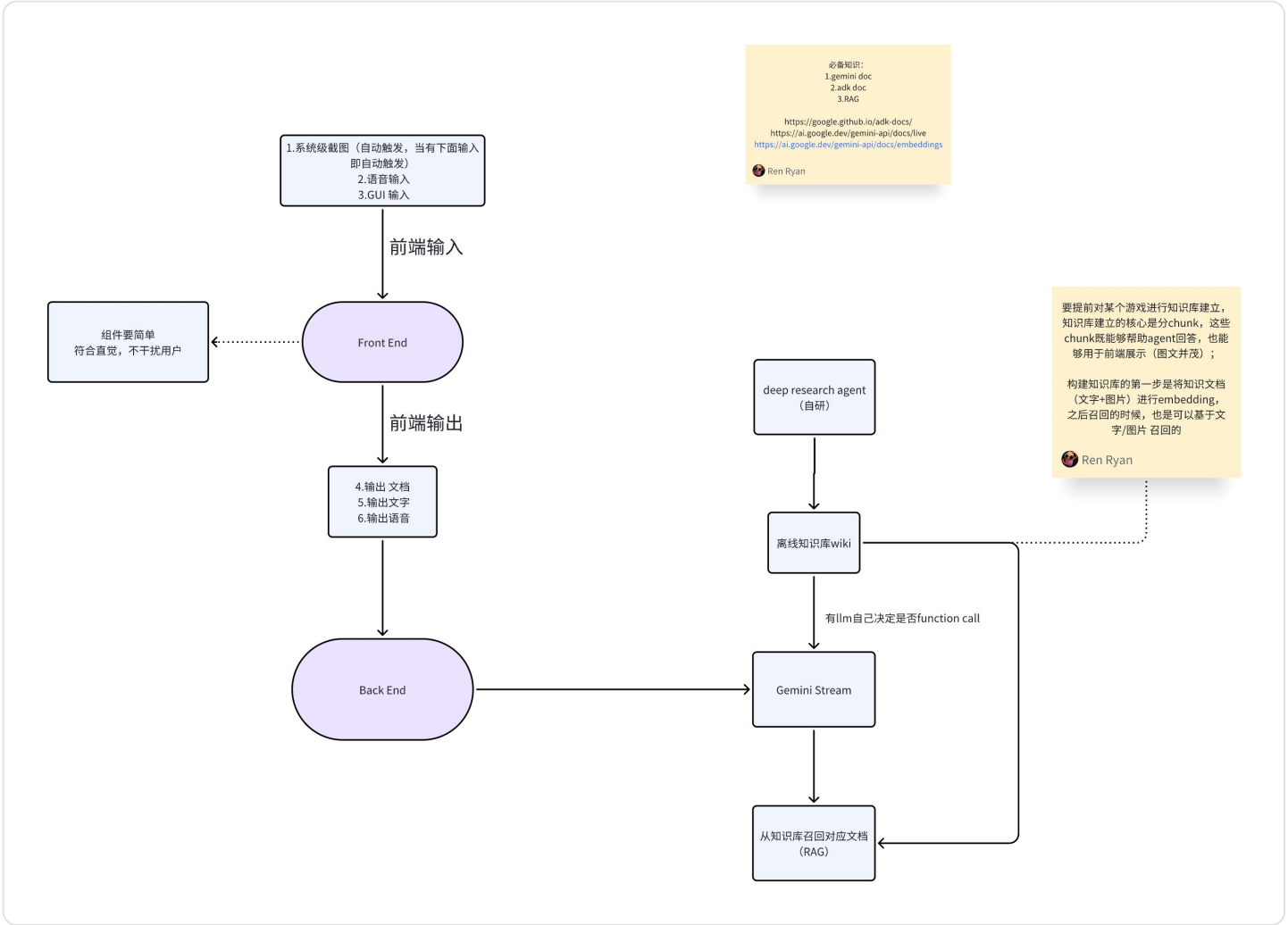
热键 / 语音 → 场景识别 → Wiki 检索 → LLM 摘要 → 浮窗文字 + TTS 播报

> 玩家：按 Ctrl+Shift+X，说“怎么打火龙？”
> 助手：识别《艾尔登法环》+OCR 得到“火龙”，检索攻略，播报“用寒冰武器，第二阶段躲龙息”并在浮窗显示要点列表。

6. 核心价值公式

`即时场景识别` + `多语言知识整合` + `AI 精炼呈现` + `语音/浮窗无中断交互`
= **减少搜索跳转时间 × 提高攻略命中率 × 保护沉浸体验**

> 以上概述汇总了产品定位、用户痛点、MVP 必备功能、后续扩展方向及关键交互链路，可作为后续团队讨论和优先级排序的快速参考。



产品需求文档

1. 当前玩家的典型痛点

- **频繁切屏查攻略，破坏沉浸感**：许多游戏的机制复杂、信息庞杂，玩家常常需要查阅Wiki或攻略才能顺利进行。目前玩家通常需要按Alt+Tab切出游戏，在浏览器中搜索攻略，这种频繁切换不仅耗费时间，还会导致游戏卡顿或中断，极大破坏游戏沉浸体验。
- **攻略信息分散且检索低效**：互联网上的游戏资料覆盖面广但零散，玩家为找到所需信息往往要在多个网页、论坛甚至视频间反复横跳。例如在开放世界或生存类游戏中，装备升级所需材料往往要翻阅长篇Wiki列表，游戏内提示不足导致玩家不得不停下来翻找资料。这种低效检索过程既耗时又容易错过关键情报。
- **缺乏内置指南，新手负担重**：不少游戏缺少完善的新手引导或任务日志（如《艾尔登法环》无任务日志），玩家容易遗忘目标和线索。他们常为记住剧情和任务进度而做笔记或频繁查阅多个Wiki页面。缺乏官方指导使得新手玩家上手门槛高、体验断裂。
- **跨语言障碍**：对于中国玩家而言，许多最新游戏资料首先出现在英文社区或Wiki上，而英文不熟练会造成获取攻略困难；反之亦然，英文玩家在玩中文游戏或查中文资料时也面临语言障碍。当前没有便捷途径在游戏中实时翻译并获取他语攻略，玩家可能错失重要信息。

- **实时互动不便**：游戏过程中手忙脚乱时，玩家很难腾出手来打字搜索，更别提阅读长篇攻略。一些紧张的实时场景下（如BOSS战或多人对战），切屏查攻略几乎不可行，玩家只能凭记忆或猜测硬撑，体验受限。

2. 产品愿景与目标用户画像

产品愿景：打造一款“游戏内的智慧军师”，让玩家在不离开游戏画面的前提下即可获得所需的攻略知识和实时指导。通过AI赋能，这款浮窗助手将深度融合多语言Wiki知识库与游戏现场情境，为玩家提供贴身、有求必应的帮助。我们的目标是让查攻略这件事像询问队友一样自然顺畅，帮助各层次玩家提升游戏体验。无论是解谜卡关、挑战高难度副本，还是寻找隐藏要素，都能在不破坏沉浸感的情况下快速得到答案或提示。

目标用户画像：本产品面向广泛的PC游戏玩家群体，尤其是有下列特征的用户：

- **硬核探索者**：热爱开放世界、角色扮演、沙盒生存等复杂游戏，喜欢探索但讨厌为了查每个细节反复切出游戏。他们需要一个随时提供资料的助手来支撑他们的探险。
- **进阶提升者**：追求竞技和优化，如MOBA、FPS、RTS玩家，希望随时了解版本攻略、装备配置或策略细节，边玩边学，迅速缩短与高玩之间的差距。
- **休闲时间有限者**：工作学业繁忙的轻度玩家，遇到卡关希望迅速得到指引而非耗费大量时间自行搜索攻略。他们重视便捷和效率，希望游戏过程顺畅不被困难卡死。
- **跨语言玩家**：玩海外游戏的中国玩家或玩国服游戏的海外玩家。他们需要消除语言障碍，在游戏中直接获取母语攻略信息，避免因语言问题错过乐趣。
- **攻略依赖者**：习惯参考Wiki、攻略网站的玩家（例如RuneScape玩家需要随时查Wiki），希望攻略信息能够更方便地与游戏融合，不再需要频繁在外部浏览器中翻找。

总体而言，本产品服务于**所有希望在游戏过程中即时获取知识支援的PC玩家**，帮助他们省去切屏之苦，提升游戏乐趣和效率。

3. 产品核心能力清单及用户价值

本产品将具备一系列核心能力，支撑上述愿景的实现。每项能力及其背后的逻辑和用户价值说明如下：

- **游戏场景识别（上下文感知）**：通过读取当前**游戏窗口标题**或对游戏画面进行OCR文字识别，自动判定玩家所处的游戏及场景。例如检测到当前游戏是《塞尔达传说》且屏幕上出现“林克的冒险”任务标题，则判断上下文为该任务。【逻辑】利用窗口句柄获取游戏名称，调用OCR提取屏幕中的关键文本（如任务名称、地点、NPC名字），结合预置的游戏知识库匹配相应条目。【用户价值】无需玩家手动输入游戏或关卡信息，系统**自动理解当前情境**并提供相关攻略，**节省操作时间**，让帮助更加精准契合当下场景。它将玩家从手动查找的负担中解放出来，避免错误上下文导致的无效信息获取。

- **悬浮窗Overlay实时呈现**：以浮窗形式将查询结果叠加显示在游戏界面顶层。【逻辑】使用操作系统Overlay技术渲染一个半透明窗口，确保显示内容不遮挡游戏关键区域，可随时召出或隐藏。【用户价值】玩家**不必离开游戏画面**即可查看攻略或提示，实现边玩边看。与传统浏览器窗口不同，浮窗轻量不扰乱游戏，保证**沉浸体验**的连续。玩家能像阅读游戏内置信息一样自然地查看攻略，大大降低因切屏带来的分心和性能问题。

- **多语言Wiki知识集成**：整合中文和英文的Wiki站点及攻略资料库作为主要信息来源。【逻辑】预先对接知名游戏Wiki（包含中文攻略百科和英文Wiki/Fandom等），建立统一的搜索与提取接口。针对用户查询，系统可并行搜索两种语言的资料，并根据用户语言偏好选择最佳答案；必要时进行机器翻译。【用户价值】玩家**获得更全面的资料来源**：中文玩家能获取英文Wiki的详实攻略并自动翻译，英文玩家也可参考中文攻略的独特见解。多语言支持确保**信息覆盖率最大化**，减少“查无此攻略”的情况，并消除语言障碍带来的困扰。

- **深度搜索与智能检索**：具备强大的搜索能力，能够针对玩家的问题进行深度检索和信息抽取。【逻辑】内置定制的搜索算法或利用搜索API，在识别游戏上下文后，将相关关键词（游戏名+主题）投入检索，爬取Wiki页面或攻略站内容，并通过算法提取可能的答案段落。此外，借助LLM的推理能力，可以多轮检索：如第一次结果不完整，则根据上下文继续自动搜索关联问题。【用户价值】保证玩家问到的问题**都有答案可寻**，包括复杂问题的分步解答。深度搜索减少玩家自己搜索的反复试错，提升答案准确率，让攻略获取**一步到位**，提升用户对助手的信赖。

- **AI知识问答与摘要**：内置大型语言模型（如即将接入的多模态AI模型 *Gemini 2.5 Flash*），能够将检索到的攻略信息进行理解加工，生成简明有用的回答或指导步骤。【逻辑】LLM接收用户的自然语言提问以及检索到的资料片段，经过分析后生成贴合问题的回答。对于直接问答型问题，模型会从知识库中总结关键点；对于流程型问题，则整理出**步骤式指导**。模型还可依据上下文调整回答风格（例如提供提示而非直接剧透）。【用户价值】玩家无需自行阅读长篇攻略再总结，**直接获得凝练答案**。这一能力让指导**更直接高效**：针对 “如何打败某Boss” 这样的提问，助手能够综合各路攻略要点，提供技能、弱点、站位等关键建议，省去玩家自行归纳的时间。借助AI智能，回答还可以更人性化和场景相关，提升互动体验。

- **语音交互（输入和输出）**：支持语音指令查询和语音朗读回答，打造全程可语音对话的体验。【逻辑】集成语音识别（ASR）模块，将玩家的语音问题转换为文本；利用TTS（文本转语音）技术，将AI的答案转换为自然语音播放给玩家听。【用户价值】实现**免打字、免阅读**的操作，大幅降低玩家与助手交互的成本。在激烈战斗或双手繁忙时，玩家只需说出疑问，几秒内就能**听到**答案，而不必分神去阅读浮窗文字。这种语音驱动的互动方式使助手更像一个随行的“智囊搭档”，玩家能够始终眼睛不离游戏画面，保持专注。此外，对于不擅长打字的用户或需要查询复杂名称（如怪物名带有特殊拼写），语音输入提供了更便捷的方式。

- **对话式多轮交互**：支持持续的上下文对话，能够记住玩家刚才的问题和游戏情境，实现多轮问答。【逻辑】每次交互时，系统都会将对话历史和当前游戏状态一起交给LLM，使其理解上下文连续性。用户可在首次提问后进行追问或澄清，例如“再详细一点”或“接下来去哪？”之类的后续问题，助手将结合前一问的上下文继续作答。【用户价值】提供更**自然灵活**的使用体验，仿佛与一个懂游戏的好友交谈。玩家不必每次都重复背景，**连续提问**能挖掘更深层的信息（例如先问Boss弱点，再问推荐装备，再问战术），满足复杂场景下的逐步指导需求。这种对话能力也帮助减少一次性信息过载，让玩家可以按需深入了解。
- **上下文引导与推荐**：在用户没有主动提问时，助手可根据检测到的游戏情境**主动提供适当的引导或提示**。【逻辑】结合场景识别结果和内置规则，判断玩家可能需要的帮助：例如侦测到玩家多次在同一Boss战失败，则浮窗闪烁提示“需要打法建议吗？”；或识别到玩家进入隐藏区域，则推送一条“此区域有隐藏宝箱，是否查看获取方法？”的通知。推荐内容链接至对应攻略页面或由AI直接给出简要提示。【用户价值】这种**前瞻式关怀**确保玩家即使忘记求助，系统也能雪中送炭，降低挫败感。不过助手只在关键时机才轻量提示，不喧宾夺主，保证游戏挑战的乐趣仍由玩家掌控。

上述核心能力相辅相成：从**感知游戏情境**到**搜索整合知识**，再到**智能产出回答并以浮窗/语音呈现**，共同构建出一个贴心高效的游戏内顾问系统。这些能力的最终价值在于让玩家**随时随地获取高质量攻略**，减少挫折，提升游戏体验，同时不牺牲游戏的沉浸和挑战乐趣。

4. MVP必备功能及必要性

为在早期验证产品价值并满足核心用户需求，MVP版本将重点实现以下功能。这些功能是产品的基础，缺一不可：

1. **游戏识别与Wiki检索**：MVP必须能够识别玩家当前玩的游戏，并基于游戏名称搜索对应的Wiki攻略内容。【必要性】若无法识别游戏，助手将无法定位正确的知识库，回复可能风马牛不相及。通过窗口标题至少可确定游戏Wiki站点，例如识别当前是《Minecraft》则连接其攻略Wiki，实现**基础的内容相关性**。没有这一功能，用户每次都得手动声明游戏，体验将非常糟糕。
2. **基础悬浮窗显示**：实现可呼出的游戏内浮窗，显示文本攻略答案。【必要性】浮窗是整个产品存在形式，没有它，AI回答就无法在游戏中呈现，用户仍需切屏查看结果，违背初衷。MVP将提供**可自由开关的浮窗UI**，支持基本的文本显示和简单排版。这保证玩家能在游戏画面上直接阅读答案或攻略段落，完成“不离开游戏获取信息”的核心诉求。
3. **查询触发机制（热键）**：支持用户通过自定义热键唤醒助手并输入问题（文本或语音）。【必要性】快捷呼出方式是MVP最低要求的交互入口。热键相比鼠标点击更高效，也不会干扰游戏操作。没有便捷触发，用户很难在紧急时刻快速提问。MVP将默认设置如 `Ctrl+Shift+X` 这类组合键唤醒助手，使玩家**随时快速提问**成为可能。
4. **跨语言知识库接入**：在MVP中初步打通中文和英文攻略源，至少实现根据用户语言偏好获取相应语言的Wiki内容。【必要性】双语支持直接决定了产品的受众广度和信息丰富度。若仅支持单一语言，MVP将无法满足另一语种用户，更无法体现跨语言获取更详实资料的价值。在MVP阶段，我们将至少接入中文维基和英文Wiki/Fandom等源，让**中英用户都能用**，并验证跨语言检索的效果。

5. **AI问答生成**：集成基础的大语言模型，能够将Wiki内容加工为简洁回答或步骤。【必要性】这是产品智能性的体现，也是与简单浏览器查询的分水岭。没有AI总结，助手只能直接展示Wiki原文，用户仍需自行阅读理解，体验提升有限。通过LLM生成简明答案，MVP即可**解决用户快速获取要点**的核心需求。例如玩家问“这个Boss怎么打”，系统直接给出策略要点，比起甩给用户整篇攻略更具价值。此功能验证了AI辅助攻略的可行性。
6. **语音输出**：实现文本答案的语音朗读功能（TTS）。【必要性】语音朗读在MVP中优先于语音输入，因为输出端对用户感知价值更直接。很多场景下玩家**听到**提示比读文字更方便（如专注操作时）。没有语音播报，助手的实时性和便利性下降一个等级。通过TTS，哪怕MVP阶段语音识别输入未完善，玩家也能享受**耳边攻略**的体验，提高产品差异化竞争力。
7. **基本语音识别**（可选的MVP加分项）：如果条件允许，MVP将集成有限的语音识别，用于简单指令查询（例如玩家可长按热键说话进行提问）。【必要性】语音输入进一步降低交互门槛，但其技术复杂度较高。若开发资源有限，此项可在MVP中基础实现（如调用现有库实现听写），为后续完善语音对话打下基础。其价值在于验证语音查询的实用性，并收集用户语音使用反馈。
8. **基本上下文对话能力**：MVP将支持至少一次追问的上下文记忆（例如用户提问后可在不重复游戏名情况下进一步提一个相关问题）。【必要性】虽然全面的多轮对话可放宽到后续版本，但**基本的上下文承接**在MVP即可增加产品黏性和智能感。没有任何对话记忆，用户每问一次都需重新描述背景，体验生硬。实现简易的上一问记忆，验证多轮交互的价值，为将来复杂对话铺路。

上述功能共同确保了MVP版本可以支撑**“游戏内快捷问答”**的核心用例：玩家按下热键，提出问题，系统识别游戏并搜索Wiki，由AI生成答案，通过浮窗文本和语音同步反馈给玩家。这一流程涵盖了识别、搜索、生成、呈现的闭环，已可以解决最主要的用户痛点。因此，这些功能是MVP中**不可或缺**的基石。

5. 非MVP可拓展功能列表

在MVP之外，本产品有大量拓展空间，可以在后续迭代中逐步实现以下功能，以增强体验并扩大使用场景：

- **高级多模态识别**：在基本OCR文字识别外，进一步引入计算机视觉模型识别游戏画面中的环境、物体和事件。例如自动识别当前屏幕上的NPC类型、怪物名称或物品图标，不依赖文本提示也能推断玩家所处情境。这将允许助手在没有明确文字的情况下也能提供帮助（比如“看到屏幕上出现一只特殊龙，助手识别出是火龙并提醒其弱点”）。该功能需要训练特定游戏的视觉模型，可逐步覆盖热门游戏。
- **玩家状态感知与个性化**：深度读取玩家的游戏状态数据，实现定制化建议。例如通过OCR/视觉分析玩家角色的当前等级、装备、技能点分配，甚至读取游戏日志/API获取玩家任务进度，然后给出**因人而异**的建议（“你当前敏捷较低，下一步建议点敏捷提高闪避”）。这相当于**私人教练**功能，让攻略建议更加符合玩家实际情况，而非千篇一律。实现方式可能包括与游戏插件集成或机器学习模型训练，需要视游戏支持情况逐步落地。

- **自动笔记和任务日志**：为玩家提供游戏笔记功能。助手可以在后台**自动记录**玩家经历的重要信息，例如剧情线索、NPC对话要点、获得的提示道具等。当玩家一段时间后回到游戏或忘记线索时，可询问助手“我之前做到哪里了？”助手能够依据记录提醒玩家下一步要做什么。这一功能解决了游戏内无任务日志的痛点，充当**第二记忆**，让玩家不再需要自行记笔记。后续版本可提供笔记的编辑和展示界面，供玩家查看历史记录。
- **主动式攻略推送**：在不打扰玩家的前提下，增加智能事件监听和攻略推送机制。例如检测到玩家多次在某一关卡失败，系统可以主动在旁边显示“需要帮助吗？”按钮；或者玩家首次进入隐藏关卡时，助手自动弹出简短提示“该区域隐藏Boss较难，做好准备”。此外可拓展**日常提醒**（如长时间游戏后提示休息）等功能。这些推送以**可配置**方式提供，确保不影响不需要帮助的玩家。
- **跨平台和移动端支持**：除了PC端浮窗，后续可探索移动设备作为辅助屏幕显示攻略。例如开发手机App，与PC端联动，在手机上显示与游戏同步的攻略信息，或通过手机麦克风进行语音提问。这样即使玩家只有单显示器，也能利用第二屏幕查看攻略，实现**多屏协同**的体验。
- **更丰富的知识源和媒体**：逐步扩充信息来源，接入论坛精选帖、YouTube攻略视频（在浮窗中播放关键片段），以及玩家社区的问答内容。未来甚至可与游戏官方数据库对接获取精准数据。例如装备掉落率等硬数据，也可以通过接口获取并呈现。此外，引入图片模式，当玩家询问地图位置时，助手可在浮窗显示标记了位置的地图截图，提高答案直观性。
- **自定义个性和社交共享**：允许用户定制助手的语音和风格（如选择不同声优的语音，或语气更幽默/严肃）。同时增加分享功能，一键将助手提供的攻略或笔记分享到社交平台或社区，方便玩家炫耀成就或帮助他人。还可引入**社区共创**要素，玩家可提交纠正或补充的攻略信息，经过审核后丰富助手的知识库，使其不断成长。
- **性能优化与离线模式**：针对后续AI模型可能对本地硬件要求高的问题（如N卡G-Assist要求12GB以上显存），提供云端推理选项和本地轻量模式切换。甚至在未来支持**离线基础问答**（预先下载常用攻略数据），让玩家在网络不佳时也能获得有限的帮助。性能调优保障助手长期运行对游戏帧率影响最小化，提升用户信赖。

以上拓展功能将按轻重缓急逐步实现。它们旨在**深化产品价值主张**：从最初的问答助手进化为全方位的游戏AI伙伴。随着多模态感知、个性化和社区内容的加入，产品将越来越聪明、好用，覆盖更多场景，吸引更广泛的用户人群。

6. 主要使用场景、用户行为路径与交互细节

本章节通过典型场景演示玩家如何与AI浮窗助手交互，以及浮窗、语音、搜索/OCR等功能在场景链路中的协同工作。

示例：开放世界游戏中AI浮窗助手的问答场景。右侧浮窗显示了玩家提问“早期最佳武器是什么？需要哪些材料？”，助手实时给出回答并语音播报，帮助玩家在不暂停游戏的情况下获取攻略信息。

场景1：卡关问答，边玩边查攻略

玩家在剧情向单机游戏中遭遇强力Boss数次失败，于是按下预设热键呼出浮窗助手，同时通过语音问：“这个Boss怎么打？” 系统识别到当前窗口是《Elden Ring》（艾尔登法环），OCR检测到屏幕

下方Boss名称为“烈焰吞噬者”。助手据此自动在后台检索艾尔登法环Wiki中有关“烈焰吞噬者”的攻略。大型语言模型读取攻略要点，生成精炼回答：例如“**攻略要点**：烈焰吞噬者畏惧寒冷武器，可以使用寒冰属性攻击；第二阶段注意躲避范围火焰，并利用柱子掩护。建议等级50以上再挑战。” 此答案随即在浮窗中显示，*TTS语音*同步播报。整个过程玩家**无需暂停游戏或切屏**：他们一边操控角色走位，一边从耳机里听到提示。【交互细节】此场景中浮窗以**QA对话框**形式出现，问题由用户语音输入后自动转成文字显示，答案文本实时滚动显示。玩家听完后，直接根据提示调整策略攻击Boss，成功过关。若未成功，玩家还可以按热键再次唤出助手，追问“有没有更简单的打法？”——由于助手保留了上一轮上下文，它明白用户仍在问同一Boss战，因而提供例如“利用召唤灵体吸引仇恨”等进一步建议。整个交互过程中，语音输入+输出使得玩家**手眼始终专注于游戏**，而AI则如影随形提供知识支援。

场景2：解谜寻路，OCR协助搜索

玩家在解谜冒险游戏中来到一处石碑，上面刻着一段生僻的古代文字线索。玩家长按语音键，说出指令：“这段文字是什么意思？”，同时用鼠标指向石碑按下截屏键。助手捕捉当前屏幕图像并执行OCR，成功识别出石碑文字内容。系统判断游戏为《古墓丽影》系列，根据识别的谜语文本在中英文攻略站搜索对应谜题解析。由于这段文字在中文Wiki上没有现成翻译，助手找到了英文攻略论坛中关于该谜题的讨论帖子。接着，LLM将英文讨论内容翻译总结为中文提示：“**谜题提示**：石碑提到的‘逐日者’其实指向地下密室。你需要在正午时让阳光照射祭坛才能打开密室入口。” 浮窗以对话形式呈现这段提示，并配以*小地图图示*（助手从Wiki图片中抓取相关谜题场景图）。玩家据此恍然大悟，顺利解开谜题。**交互细节**：在此过程中，OCR提取文字使玩家不用手动输入难记的字符，搜索和翻译功能让跨语言攻略得以利用。浮窗在显示文本答案的同时，还可展示一张相关小图，玩家点击图像可放大细看。这种多模态协同确保即便是复杂的文字谜题，玩家也能**一站式**获得解释。

场景3：策略优化，实时咨询

一名RTS（即时战略）玩家希望改进自己的开局策略。在对战空闲间隙，他打开助手（这次通过快捷键并键入问题，因为周围环境嘈杂不便说话）：“快速提升经济的开局 build order 是什么？” 助手检测游戏为《星际争霸》，结合玩家当前使用的种族OCR识别（例如界面上有虫族标志），自动检索星际争霸虫族的快速经济开局策略。很快，浮窗显示一个**步骤列表**：“1. 9人口时拍下一个吓虫（提供经济加成）；2. 然后迅速升本；3. 在XX秒时开始孵化场……” 并附上简短解释。玩家迅速浏览步骤，在下一局开场时按照助手给的顺序执行，明显感觉资源提升更快。**交互细节**：该场景突出助手的**步骤式指导**能力。浮窗特地设计为分条列表以方便阅读，语音播报则在每局开始前提前朗读以方便玩家记忆。在玩家执行过程中，如有遗忘可以随时再唤出复习下一步。这体现了AI助手在竞技游戏中**充当教练**的角色，无需暂停游戏即可获得专业攻略建议。

场景4：跨语言查询与翻译

一位中国玩家正在玩一款全球同步上线的新游戏，但该游戏的详细攻略主要散布在英文社区。玩家在游戏中遇到不懂的机制，例如一个属性效果不明，于是用中文提问：“**‘Vorpal Damage’是什么效果？**”。助手识别游戏为某奇幻RPG，自动搜索英文Wiki。Wiki上有对此属性的解释说明，但没有中文资料。助手遂将英文说明通过LLM翻译并简化为中文：“*Vorpal伤害*：一种会心伤害效果，攻击有小概率直接斩首普通敌人，对首领无效。” 并补充了出处来源以供参考链接。玩家立刻理解了这个属性的意义，避免了望文生义的误解。**交互细节**：助手根据玩家的系统语言/提问语言决定回答语言，这里玩家用中文问，尽管知识来自英文Wiki，也**自动输出中文**。浮窗还提供一个按钮“查看原文”，如玩家点击，会在浮窗中打开英文Wiki原文片段供深入阅读。这种跨语言能力让玩家无需离开游戏去求助翻译工具，在同一个对话中完成提问、获取翻译解释的闭环。

场景5：语音伴侣模式

在一款大型开放世界RPG中，玩家长时间探索可能感到孤单。助手可以提供一种伴侣模式：玩家不需要特定疑问，也可以和助手闲聊当前进度或寻求建议。例如玩家说：“我们下一步该去哪探索？”，助手基于玩家目前地图区域和主线进度，从知识库中找到该阶段可以探索的支线任务或隐藏要素，于是语音回答：“你刚完成主线第三章，现在可以去北方雪山找隐藏Boss，也可以返回城镇升级装备。我建议先升级装备以应对接下来的难关。” 玩家觉得有道理，跟随建议行动。**交互细节**：在此模式中，助手更像**游戏内NPC**，随时待命提供建议和世界背景解说。这利用了多轮对话和上下文理解，让交互更加自然。不过为了避免剧透，助手遵循设定：除非玩家明确要求，不主动提及尚未触发的剧情。这确保交流既有帮助又不破坏故事体验（沉浸保护的一种体现）。

综上所述，这些场景展示了AI浮窗助手在**单人剧情、解谜探险、竞技对战、跨语言游玩**等不同情境下的应用方式。无论是被动提问还是主动辅助，助手都通过浮窗、语音、OCR、深度搜索等技术的配合，实现了**“游戏中即时获取所需信息”**的体验革新。在交互细节上，我们注重：**快捷召唤**（热键/语音唤醒）、**非侵入性显示**（小窗、不遮挡UI）、**内容渐进呈现**（支持按步骤、查看原文）、**多模态反馈**（文字+语音+图片）等，使得用户与助手的交流高效顺畅，不影响游戏正常进行。

7. 特别注意事项

在产品设计和实现过程中，需要关注以下特殊问题，以确保跨语言、多模态交互顺利，并最大程度保护玩家的沉浸式体验：

- **跨语言支持与术语一致**：由于涉及中英文双语环境，需建立游戏术语对照表与翻译机制，确保助手回答时使用玩家熟悉的语言和词汇。例如同一物品在中英文Wiki名称不同，助手必须识别对应关系并统一成玩家所用语言。还有对于玩家自行提问中夹杂的外文专有名词，要正确识别并检索。确保中英信息源内容在融合输出时**语义一致，不产生歧义**。同时，界面文本和语音播报都应根据用户语言偏好自动切换。
- **多模态交互流程与性能**：语音识别、OCR、图像分析等多模态功能在提升体验的同时，也带来性能开销和复杂度。必须注意优化处理流程，尽量**异步**执行OCR/搜索等操作，防止阻塞游戏进程。对于GPU占用高的模型推理，可以考虑在GPU空闲时段或采用云端推理，以免影响游戏帧率。另外，要制定优先级：如同时触发语音输入和OCR解析时，如何调度顺序以及出现冲突时的回退策略（例如OCR失败时提示用户改用文字描述）。良好的错误处理和反馈机制很重要：若OCR识别不出文本，浮窗应提示玩家调整画面或手动输入关键字，而不是默默失败。
- **沉浸体验保护**：设计上要平衡提供信息与保护游戏体验之间的度。首先，浮窗UI风格应尽量简洁美观，融入游戏氛围，避免突兀分散注意。支持**用户自定义**浮窗的位置、大小、透明度，以适配不同游戏界面（防止遮挡血条、小地图等重要元素）。其次，语音播报音量和语气需考量游戏环境：在剧情对话时助手不应喧宾夺主，必要时可延迟回答或使用较低音量/不同声线提示，以免扰乱玩家情绪。再次，**防剧透机制**要有：默认回答只针对玩家提出的问题，不主动泄露剧情走向。例如当玩家问“下一步去哪”时，若涉及剧情反转，应先提示是否继续深度讲解，以尊重玩家选择。最后，确保助手的**存在感可控**——玩家不需要时它安静隐藏，不主动弹出广告式的信息干扰游戏。总之，任何功能的添加都以不破坏沉浸为前提，做玩家的幕后支持者。
- **隐私与安全**：语音输入和屏幕OCR过程中涉及用户隐私和游戏画面的捕捉，必须严格保障数据安全。语音识别只在本地或可信云端执行，不会将录音长期存储；OCR截取的屏幕图像应仅用于瞬时解析，且不上传未经处理的完整截图。对于联网查询，要注意过滤任何可能的恶意脚本或不良信息，输出内容需经过安全审查，避免将不当内容带入游戏环境。特别是在多人游戏中，要明确助手只是一款客户端工具，不具备透视、修改游戏的数据等**影响游戏公平**的行为，以免被归类为作弊工具。产品需与各大游戏的用户协议保持一致，确保合法合规运行。
- **兼容性与易用性**：考虑到不同游戏的运行模式（全屏、无边框窗口等），浮窗功能需适配各种渲染模式，提供可靠的显示效果。同时注意与各类反作弊系统的兼容：浮窗读取屏幕和进程名称的行为可能被某些反作弊误判，因此需要与厂商沟通或采取安全模式（例如仅OCR静态画面而不注入进程）。用户体验方面，新用户上手应有引导教程，例如首次运行时指导其设置热键、授予OCR权限等。语音交互也应提供反馈机制，比如在助手侦听时显示麦克风图标、录音时间过长时提醒用户收尾，避免操作迷惑。通过**细节打磨**，让多模态交互变得直观可信。

综上，AI游戏浮窗助手的设计开发既要敢于利用前沿AI技术为玩家创造价值，也需慎重考虑实际使用中的各种细节和限制。只有在跨语言准确性、多模态性能以及用户沉浸感方面都做到**精益求精**，这款产品才能真正赢得玩家信赖，成为游戏体验的有益拓展而非干扰。通过稳健的MVP落地和持续的功能演进，我们有信心实现产品愿景：让每一位玩家都能拥有一个贴心聪明的游戏内AI助手，在广袤的游戏世界中畅行无阻。

AI 升级开发计划

当前实现与产品目标的差距分析 (Gap Analysis)

当前 `agent` 分支的代码功能与产品目标相比存在多方面差距：

- **AI 搜索问答能力缺失**：现有代码仅支持根据游戏名称加载预设网页内容或用户输入关键词检索维基页面，没有集成任何大型语言模型 (LLM) 来智能回答问题。这意味着没有上下文推理或多轮对话能力，无法满足产品目标1中所要求的 Gemini 2.5 Pro 与 Flash 模型驱动的智能问答。当前每次查询都需要用户手动输入关键词，无法自动理解游戏情境或问题意图。
- **语音交互缺失**：现实现中没有任何麦克风语音输入或TTS语音输出功能。用户只能通过热键弹出文本输入框交互；系统回答也仅以浮窗形式展示网页内容，没有语音播报。产品目标2要求集成Gemini TTS进行语音播报，并支持“只展示”或“语音+展示”模式，但目前代码的设置和UI中没有语音相关配置。因此需要新增语音输入的捕获和识别，以及语音合成输出的模块。
- **上下文感知能力不足**：当前系统只能获取前台窗口标题来判断游戏名称，用于选择预设URL或检索模板。没有对游戏画面内容的分析流程，也未使用OCR技术提取游戏界面文本。产品目标3要求能截屏并OCR分析当前游戏页面，以理解游戏状态或玩家所处情景，再由AI自动回答。这一整套页面上下文识别链路在现有实现中完全缺失，需要从零开发。
- **多轮对话和记忆缺失**：目前每次用户按下热键触发的查询都是独立的，程序不保存先前问答的内容。产品目标1提到的多轮对话和上下文推理能力意味着AI需要记住对话历史并据此理解后续提问。现有代码没有任何会话存储或上下文传递机制，每次都重新从游戏名和用户关键词开始，缺乏对话记忆支撑。
- **多语言多模态支持不足**：现有功能主要围绕中文游戏维基（如游戏名和维基URL包含中文）。没有针对双语互动进行特殊处理。目标要求系统能用中英双语与用户交流，并处理图像（截图）等多模态输入。这需要确保底层模型Gemini支持中英双语，并能处理图像输入或OCR结果。同时UI层面需要考虑不同语言文本的显示，以及在TTS播报时选择合适语言的语音。当前实现未考虑这些因素。
- **UI 和配置方面**：目前的UI只是一个隐藏到托盘的WPF应用窗口，提供热键设置和固定大小位置的WebView浮窗。为满足新功能，需要扩展UI/配置：例如增加语音模式开关（“仅文本” vs “文本+语音”）、可能的麦克风权限提示或指示录音状态的UI、以及多轮对话的重置或历史查看功能等。此外，现有输出浮窗仅嵌入WebView展示网页，而AI生成答案可能需要在浮窗中以文本对话形式显示，UI呈现方式需要调整。

综上，当前实现与目标需求存在明显的技术鸿沟，需要围绕**模型集成、语音收发、多模态感知、对话管理**等方面进行升级开发。

整体方案设计：模块划分与AI能力集成

为实现上述目标，本计划拟定基于现有WPF架构，嵌入Google Gemini模型和ADK能力的模块化方案。主要模块及其职责如下：

- **输入模块（语音/热键）**：负责捕获用户提问触发，包括全局热键监听和语音输入监听。
  - 在“仅展示”模式下：按下热键将弹出文本输入框（类似当前 `Prompt.ShowDialogAsync`）供用户键入问题。
  - 在“语音+展示”模式下：按下热键将启动麦克风录音，获取用户语音提问。录音过程中可在界面上提供指示（如浮窗或托盘提示正在倾听）。录音结束（可通过再次按热键或检测静音停止）后，将音频传递给下游处理。
  - 语音识别：利用Google云语音识别（Speech-to-Text）API或本地引擎，将捕获的语音转写为文本问句。**注**：语音识别可通过Google Cloud .NET SDK直接调用，也可作为ADK工具集成（ADK支持调用Google Cloud服务）。鉴于开发效率，可先使用Google的预训练ASR服务确保中英双语识别准确。转写完成后，得到用户的文本问题。
- **游戏上下文感知模块**：负责获取当前游戏环境信息，辅助AI理解提问背景。
  - **截图捕获**：利用Win32 API获取当前游戏窗口句柄，然后截取该窗口画面（例如通过 `PrintWindow` 或 GDI+ BitBlt）。如果窗口句柄不可得，也可退而求其次截取全屏。获得截图图像后，交由OCR识别。
  - **OCR 文本提取**：集成OCR引擎提取截图中的文字。优先考虑Google Cloud Vision API的文本识别，因其对游戏界面中文字的准确率较高，并支持中英等多语言。可通过ADK的Google Cloud集成工具调用Vision API，实现截图OCR。若需本地方案，也可使用Windows自带OCR（Windows.Media.Ocr）或Tesseract OCR，兼顾实现难度和识别效果。
  - **结构化上下文**：将OCR获取的文本（例如当前任务描述、NPC对话、地名等）与其它易得的上下文信息一起打包。其它信息包括：当前游戏名称（通过窗口标题或用户选择确定）、也可包括时间（游戏时间/系统时间）或之前QA历史中总结的游戏进度。最终形成当前轮问答的**上下文数据对象**，包含{游戏: X, 场景文本: Y, 其他…}，为AI提供额外背景。
- **AI 问答代理模块**：这是系统的智能核心，利用Google Gemini 2.5 Pro和Flash模型，通过Agent Development Kit (ADK)编排对话流程，实现搜索、推理和多轮对话。该模块可以以**Python服务**形式实现，利用ADK强大的对话管理和工具集成能力，与前端WPF通过接口通信。具体设计如下：
  - **架构**：使用Google的 **Agent Development Kit (ADK)** 创建一个定制智能体。选择Python版本的ADK进行开发，原因是Python生态对调用AI模型和云服务更成熟，ADK提供开源库便于快速构建。智能体的大致逻辑是：接收用户问句和上下文 -> 必要时调用工具(如网络搜索、OCR*****等) -> 调用LLM获取回答 -> 返回答案（并持续维护会话状态）。

- **模型集成**：在ADK中配置接入Google Gemini模型。Gemini 2.5 Pro用于主要对话生成，具备最强的上下文理解和推理能力；Gemini Flash模型可用作辅助，例如执行快速检索或简单问答以加速响应。通过ADK可以同时注册多个模型，让智能体根据任务选择使用。可能的策略是：对于需要复杂推理或多模态输入的提问，调用Gemini Pro；对于简单常识问答或重复性查询，可用较轻量的Flash模型快速给出草案答案，再由Pro润色确认。这种双模型配合能兼顾响应速度和回答质量。
- **工具调用 (搜索/函数)**：借助ADK的**工具集成**功能扩展智能体能力。例如集成**网页搜索工具**：在AI判定自身知识不足时，自动触发一个受控的网络搜索（如调用游戏维基API或Google搜索API）获取相关资料，然后将结果纳入上下文供LLM综合分析。这实现了产品目标1中的“搜索问答”能力。也可以集成**游戏Wiki查询函数**：基于现有games.json映射，提供一个函数接口，AI传入游戏名和关键词参数即可返回对应Wiki页面内容摘要。通过ADK，LLM可调用这些函数型工具（类似OpenAI函数调用）来检索知识，然后继续对话。
- **上下文处理**：将前述游戏上下文数据（OCR文本等）提供给LLM。Gemini模型是多模态的，理想情况下可直接输入图像及文本。但若接口不直接支持图像输入，可退而求其次：将OCR识别出的文字描述作为系统提示的一部分，或作为函数调用结果注入对话。例如可以构建提示词：“当前游戏状态：{场景文本}。用户问：{问题}。”让模型利用这些上下文进行推理。Gemini 2.5 Pro强大的上下文理解能力和MCP协议(模型上下文协议)将确保模型充分利用这些信息。
- **多轮对话与记忆**：利用ADK的**会话和记忆**机制来实现上下文保持。ADK支持对话Session，在本地可用内存会话，在部署到云后则使用托管会话服务。开发者可通过 ADK 的 Session API 创建和管理会话，每次交互关联到同一 session_id，从而自动累积对话历史。此外，可利用ADK内建的Memory组件存储长期知识或重要事实。实现上，智能体每次回答前会将最近若干轮对话（用户问句+AI回答）作为对话历史提供给模型，或使用ADK提供的“Memory”中存储的总结。这样模型可以参考之前的内容理解省略指代的问句，满足产品目标1的多轮对话需求。
- **智能体部署接口**：为了与WPF前端交互，考虑两种方案：
  i. **本地后端服务**：将AI代理模块封装为一个本地HTTP/IPC服务。WPF在需要问答时，将用户的问题文本和上下文数据通过HTTP请求发送给本地Python服务，后者调用ADK智能体处理并返回答案文本（以及可能的富媒体内容或音频URL）。此方案调试方便，延迟低，适合单机使用。
  ii. **云端代理服务**：利用Vertex AI Agent Engine部署ADK智能体到云端。WPF前端通过Google Cloud API将请求发送到云端代理，由云端执行语音识别、OCR、LLM调用等并返回结果。这样可利用云端算力和托管会话，但需处理网络延迟和凭证配置。一人开发初期可先采用本地方案，待功能成熟再部署云端以利用云托管和更强算力。

- **输出展示模块**：负责将AI代理返回的结果以用户可感知的方式呈现，包括文本浮窗和语音播报两个部分。
  - **浮窗显示**：复用现有WPF浮窗（`Window popup`）机制来显示AI回答内容，但需要从原先加载网页改为显示AI生成的文本/富文本。可以有两种实现方案：

    i. **WebView2 加载本地HTML**：将AI的回答组装成HTML文档（例如带有简单的CSS样式和内嵌的图片/链接），然后使用 `WebView2.CoreWebView2.NavigateToString()` 或加载temp文件的方式在浮窗中渲染。这延续了原有WebView显示流程，开发成本低，并可轻松支持富文本格式和嵌入内容。

    ii. **纯WPF控件显示**：直接在浮窗窗口中加入TextBlock/FlowDocumentViewer控件显示回答文本。这样可以减少依赖，但格式定制和滚动支持需要自行处理。鉴于现有代码已经初始化了WebView2环境，采用WebView2渲染HTML更为直接。开发者可以设计一个HTML模板，将AI回答插入适当位置，例如模拟聊天气泡或卡片式答案展现，使界面更加美观易读。浮窗仍使用Topmost属性悬浮于游戏画面上方，保证用户边玩边看答案。

  - **语音播报**：在“语音+展示”模式下，调用Gemini TTS将AI回答的文本合成语音。Google Gemini TTS（或云Speech Services）支持中英双语及多种音色选择，确保AI用适合的语言和自然的声音反馈用户（例如用户用中文提问则使用中文语音）。集成方式可以通过Google Cloud Text-to-Speech API，将文本发送获取音频数据（MP3/WAV），再由前端播放。播放可使用WPF的MediaElement控件或NAudio库，实现非阻塞播放并允许音频结束事件。需注意在调用TTS时异步处理，避免阻塞UI线程。语音播报的开关由用户配置控制：如果用户选择“只展示”，则跳过TTS调用，只保留文字显示。

  - **多轮对话UI**：如果实现多轮对话，浮窗显示可以扩展为对话视图，依次展示每轮Q&A。可以在HTML模板或WPF控件中，将用户提问和AI回答以不同样式区分。若使用WebView2方案，可在每次新回答生成后，通过JavaScript或重新生成HTML的方式，附加新问答条目，保持历史可见。需同时考虑浮窗的尺寸和滚动，以容纳多轮内容。用户也应有方式重置对话（例如在托盘菜单提供“重置对话”选项，清空会话历史/Memory）。

- **配置与状态管理模块**：扩展现有settings.json配置文件和设置UI，以支持AI相关选项：

  - 添加布尔配置项如 `UseVoiceOutput`（语音播报开关）、`UseVoiceInput`（是否默认语音输入模式）等，由用户通过主窗口UI或托盘菜单修改。保存配置后，在应用启动时加载，在相应模块初始化时读取这些设置。例如，决定是否注册/启用麦克风监听，是否对回答启用TTS。

  - 管理API密钥或凭证：调用Google各API需要凭证，单人开发可将API密钥存于本地配置（加密存储），在调用时读取。亦可加入简单UI让用户粘贴其Google Cloud密钥。注意保护敏感信息。

  - 运行时状态管理：跟踪当前会话ID（若使用ADK Session）、当前是否在录音中等状态，以便模块间协同。例如，当语音识别尚未完成时暂缓触发AI查询；当新回答正在朗读时，如果用户又提了新问句，则可以打断上一语音播放等。这些状态和逻辑需在主程序中妥善处理。

以上模块各司其职，通过**清晰的接口**衔接：输入模块获取的文本和截图上下文进入AI模块，AI模块输出回答文本/音频，输出模块呈现给用户。通过这种分层解耦，单个开发者可以逐步实现和调试每个功能模块。

详细对话流程（AI链路）设计

结合上述模块，下面以用户完整交互流程说明系统工作链路，展示各部分如何协同，实现产品目标4要求的“完整流程”：

1. **用户触发**：用户通过**全局热键**触发GameFloaty。当启用了语音模式时，此时应用开始录音用户的问题；若仅文本模式，则直接跳转第3步。
   - *实现细节*：在 `WndProc` 监听到热键消息时，检查配置决定走语音还是文本分支。语音模式下，调用麦克风录制函数开始收集音频数据，并在UI上给予提示（如托盘图标闪动或浮窗显示“Listening...”）。录音结束的判断方式可以是用户再次按下相同热键键或设定最大录音时长等。
2. **语音转文本**：录音结束后，系统获得音频流，调用 **语音识别** 服务将其转成文字问句。例如，用户问：“这个Boss有什么弱点？”。
   - *实现细节*：调用Google Cloud Speech-to-Text API，将音频数据发送并获取识别结果字符串。考虑到游戏环境下可能有背景音，需使用抗噪声的识别模型，并可根据用户偏好指定识别语言（如中文）。一旦返回结果文本，将进入下一步。若识别失败或结果信心低，可以提示用户重试或回退到文本输入方式。
3. **获取游戏上下文**：同时（或紧随其后）系统捕获当前游戏画面信息，以便提供给AI参考。
   - *实现细节*：通过前述截图+OCR模块获取文字描述。如当前屏幕显示Boss名称“炎龙”，以及Boss生命值等HUD信息，OCR结果可能提取到“Boss：炎龙\nHP：3400/5000”等。再比如当前任务描述文本等也会被OCR捕获。将这些文字与游戏名称一起打包成上下文。例如：`ContextText = "当前敌人：炎龙（HP 3400/5000）；地点：火山洞穴；当前任务：击败炎龙拯救村民"`。实际实现中，可对OCR的结果做适当清洗、截断无关信息，保留关键字以减少噪音。由于此步骤可能耗时（截图+OCR），建议与语音识别并行执行（两者独立）以减少总等待时间。
4. **生成AI请求**：构造发送给AI代理模块的请求，包含：用户提问文本、游戏上下文文本，以及会话ID（如果有多轮对话）。
   - *实现细节*：如果采用本地Python服务，可以构造一个JSON，例如：

代码块

```
{
  "session_id": "12345",
  "user_query": "这个Boss有什么弱点？",
  "game": "Monster Hunter World",
  "context": "当前敌人：炎龙 ...",
  "history": [ {"role":"user","content":"上轮问题"},
{"role":"assistant","content":"上轮回答"} ]
}
```

- 其中 `history` 可选，用于多轮时传递已有对话。前端通过HTTP POST发送此JSON到本地ADK代理的REST接口（例如 `http://localhost:5000/query`）。如果使用云端Agent，则通过SDK调用对应的API并传递这些参数（云端会话ID管理在ADK服务端）。在首次对话时，可能先请求代理创建新Session（ADK提供API创建会话），获取session_id后附在后续请求中。

5. **智能体处理**：ADK智能体接收请求后，按照预先定义的**对话链路**执行：
   a. **上下文注入**：将收到的 `context` 文本作为系统消息或提示注入对话，使LLM知晓玩家当前所处情境。
   b. **调用工具**（可选）：基于问句和上下文，智能体判断是否需要检索资料。例如如果问题涉及具体数据（Boss弱点可能在攻略帖），智能体调用“游戏Wiki检索”工具函数，查询“炎龙 弱点 Monster Hunter World”。工具返回的结果（例如维基中炎龙弱点的段落）被纳入后续提示。ADK确保模型能以函数形式请求并接收工具输出。
   c. **LLM回答**：综合用户提问、游戏上下文、以及任何工具检索结果，调用Gemini 2.5 Pro模型生成答案。由于Gemini模型强大的推理能力，它可以理解上下文（如识别出“炎龙”是Monster Hunter World中的Boss）并给出详细回答。这一过程中，如果对实时性要求高，可先尝试使用Flash模型快速生成要点，再由Pro模型参考要点输出高质量答复。ADK支持在智能体逻辑中灵活选择模型调用，从而实现双模型协同。
   d. **多轮对话维护**：ADK会将此次对话的问答内容记录到会话Memory中。如果有follow-up逻辑，例如用户提问不清晰，智能体也可以返问澄清（通过回复反问并等待用户回答，再继续），这得益于ADK对多轮交互的支持。对于本场景，通常直接给出答案。

6. **返回结果**：AI代理将生成的最终回答发送回前端。这通常是一个文本段落，可能包含结构化信息（列表、步骤）甚至带有Markdown/HTML格式以便更好呈现。另外还可以包括一些元数据，例如建议的语音语言，或引用的来源链接。

   - *实现细节*：本地服务通过HTTP响应JSON给WPF，包括 `answer_text` 字段和可选的 `sources` 或 `voice_lang` 字段。若使用云Agent，则通过SDK拿到响应对象。示例返回：

```
代码块
1  {
2    "answer_text": "炎龙怕水和冰属性攻击。建议携带水属性武器... (后略)",
3    "voice_lang": "zh"
4  }
```

   - 前端据此提取答案内容，以及语音语言选择（系统也可自行判断语言，例如检测字符串是否包含中文）。

7. **展示与播报**：前端收到AI回答后，立即在浮窗中展示文本，并根据模式决定是否语音播报：

   - *文本展示*: 如果采用WebView显示，前端将回答内容嵌入HTML模板，例如：

```
代码块
1  <div class="assistant-answer">
```

```
    炎龙怕<span style="color:blue;">水</span>和<span style="color:cyan;">冰
</span>属性攻击，...
</div>
```

- 然后调用 `webView.NavigateToString()` 将HTML加载到浮窗。由于之前已创建共享的WebView2环境，加载应较为迅速。若已经有之前QA的内容，新的内容可以追加（例如通过JS在document.body追加元素，实现对话连续显示）。
- *语音播报*: 若启用语音模式，前端同时调用TTS服务合成音频。将 `answer_text` 发送到Google TTS API，请求对应语言的语音（如voice_lang指明中文女声）。拿到音频数据后，使用音频播放器播放。同时，保持浮窗显示，以便用户可以听的同时看文字。要注意控制流程：如果下一次用户提问在上一次语音播报尚未结束时到来，应该中断当前音频（可以调用播放器的Stop方法）以优先响应新的问句，确保对话及时性。

8. **循环交互**：此后用户可继续下一轮提问。例如看到答案后问“我需要准备什么道具？”。用户可再次按下热键并讲话，系统进入新一轮流程。因已存在对话Session，智能体会将上一轮关于“炎龙弱点”的QA作为历史，用Gemini模型的上下文窗口加以参考，从而理解“什么道具”是延续询问打炎龙所需的道具。多轮对话在这种机制下实现，直到用户结束。用户可在托盘菜单选择“结束对话”或在主窗口点击“重置”按钮来清除当前会话记录，开启新的话题。

通过上述链路，实现了产品目标4描述的**“用户输入（语音或热键）→ 当前游戏上下文识别（OCR+截图）→ AI问答页面生成 → 浮窗展示 & 语音播报”**的闭环。其中各步骤充分利用了Gemini模型的能力以及ADK工具编排，实现上下文相关的智能对话。

开发分阶段计划

1. **阶段1：集成基础AI问答 (文本版)**
   目标：在无需语音和OCR的情况下，实现用户输入文本提问->AI回答->浮窗显示流程。这验证Gemini API/ADK集成和基本问答逻辑。
   实施：
   - 在WPF中新增调用后端AI的逻辑。先不启用ADK复杂功能，可直接使用Google Vertex AI大型语言模型的API（例如PaLM API或ChatGPT替代）来返回答案。
   - 修改现有 `ShowWikiWebPopup` 流程，在用户输入关键词后，不再构造URL打开网页，而是将该关键词当作问题发送给模型获取回答文本。临时可不使用games.json，假定问题本身包含游戏背景（或先硬编码上下文）。
   - 将返回的答案文本简单地显示在浮窗中（可直接设置WebView2显示一个包含答案的HTML，或用TextBlock显示纯文本）。
   - 验证模型能够返回合理答案（需要提供示例提示，如 `系统提示：你是游戏助手` 等以规避无关输出）。调整UI样式保证可读性。
   交付：用户按F键（假定热键）后，弹出文本框输入如“如何打败炎龙？”，按下回车后，浮窗显示AI返回的回答。此阶段完成后，GameFloaty已具备最小AI对话雏形（单轮，文本交互）。

2. **阶段2：引入Google ADK架构**
   目标：重构阶段1的AI调用至ADK代理，以利用多轮会话和工具扩展能力，为后续功能打基础。
   实施：
   - 在Python环境中创建ADK智能体项目，定义智能体行为：采用LLM Agent处理对话，并预置简单Memory（例如让它记忆上一句对话）。
   - 实现本地Flask/FastAPI服务，将HTTP请求映射到ADK智能体的查询接口。实现session管理：首次请求创建session并存储session_id，后续请求在HTTP层附带session_id以保持会话。
   - 在WPF前端改为调用本地ADK服务的API，而非直接LLM接口。调通基础问答（仍是文本输入输出）。
   - 测试多轮对话效果：连续输入两三个相关问题，看ADK Memory是否让模型记住上下文。如果效果不佳，可调整ADK Memory机制，或暂以简单串联prompt方式实现记忆。
     交付：架构改为前后端分离，前端负责UI和请求，后端ADK智能体负责逻辑。用户体验应与阶段1类似，但内部已为多轮对话和工具使用做好准备。
3. **阶段3：游戏上下文识别 (OCR) 集成**
   目标：让AI感知游戏当前画面信息，使回答更具针对性，实现多模态交互的一半（视觉输入）。
   实施：
   - 在WPF中实现截图功能：获取前台窗口（游戏)画面保存为图片。可使用Win32 API `GetForegroundWindow` + `PrintWindow` 实现。处理多屏幕坐标等细节，确保截图完整。
   - 集成OCR：优先尝试调用Google云Vision API。获取API密钥，使用其 .NET SDK 将截图图像发送并识别文字。处理返回结果，将文字串整理干净。若云OCR延迟较大，可选用本地OCR库做备选。
   - 将OCR文字传递给后端ADK智能体。在HTTP请求JSON中增加一个字段，如 `context="..."` 提供该文字。【注】同时需要调整智能体提示词模板，在模型调用前将context合并进去。例如ADK里设定system_prompt包含 `游戏当前信息:{context}` 。
   - 测试效果：在人为设计的场景下验证。例如在一款文字RPG游戏中，屏幕上显示任务提示“前往黑森林击败狼王”，用户只问“我现在该怎么做？”，AI应结合OCR到的“击败狼王”提示来回答下一步策略。
   - 根据测试调整：可能需要过滤OCR噪声、限制context长度（Gemini上下文长度有限）。必要时，让智能体调用工具先总结OCR内容再作答，以减少无关信息干扰。
     交付：当游戏界面有明确文字信息时，AI回答能引用或基于这些信息。比如玩家在某Boss战，AI回答会提到Boss名字和状态（源自OCR）并给出对应建议，表明AI已“看见”游戏情况。
4. **阶段4：语音输入输出集成**
   目标：实现全双工的语音对话功能，让GameFloaty具备语音助手特性。
   实施：

- **语音识别**：在前端加入麦克风录音功能。可使用NAudio库捕获麦克风PCM数据，或调用System.Media里的简易录音类。UI上在Tray图标或浮窗上显示录音指示（如一个红点或波形动画）。录音完成后，将音频数据发送到Google Speech-to-Text API识别。为了减少实现难度，可以采用 REST API 直接上传音频获取结果。配置识别语言支持中英（Cloud STT可自动检测语言或需指定）。
- **触发逻辑**：决定何时开始/停止录音。一个方案是：单击热键**按下时**开始录音，**松开时**停止（类似对讲机），这样用户体验直观。实现上需使用全局键盘钩子捕获按键状态；或者按下热键一次开始，再次按下停止（配合提示音）。此交互需清晰指引防止误用。
- **将识别文本接入流程**：当获得STT结果后，相当于替代了文本输入框的内容，然后按原流程调用OCR、发送ADK查询。需要处理识别延迟，在此期间可显示“正在识别…”状态。
- **语音合成**：调用Google Text-to-Speech服务，将AI返回的答案转语音。选择适合的音色，如中文女声或英文男声，可提供选项。API返回音频后，直接播放。注意播放时不要阻塞UI线程。可以在后台线程或利用 `MediaElement` 的事件。
- **配置开关**：在设置中加入语音输入和输出的开关，让用户能够自由选择模式。例如默认关闭语音，则热键仍调出文本框，不录音；用户开启语音后则热键改为录音模式。语音输出同理，有独立开关控制播放与否。UI上提供这些选项（复选框或下拉）。配置值存入settings.json并在启动时应用。
- **中英切换**：实现双语对话关键在于**识别和合成**两端保持一致语言。STT端可以使用auto-detect或根据游戏/用户偏好来选语言模型；TTS端可简单地通过检测answer文本首字符是中文还是拉丁字母来判断语言，然后选相应语言的语音播报。Gemini LLM本身可用任何语言回答，尽量让它跟随用户语言：这可在system提示中明确，例如“用户说什么语言你就用什么语言回答”。经测试调整确保AI不会混杂输出。
- **测试**：选取中英文不同游戏进行语音对话测试。如在英文界面游戏中用英语提问，检查识别->回答->语音是否完整。在中文游戏中中文提问亦然。根据需要调整识别参数和TTS声音。
  交付：用户无需打字即可与GameFloaty语音交流。比如用户按住F键问：“下一个目标在哪里”，松开后几秒，浮窗显示答案“按照地图，你需要去北方的山洞”,同时语音播报：“按照地图提示，你需要前往北方的山洞。”整个过程自然顺畅。

5. **阶段5：优化与完善**
   目标：在基本功能实现后，针对产品定位和用户体验进行完善优化，包括性能提升、错误处理和个性化配置。
   任务示例：
   - **优化响应速度**：引入Gemini Flash模型配合，在某些无需完整Pro模型的情况下快速返回结果。可测定平均延迟并调节策略（例如问答长度短时直接Flash，长问答Pro）。同时考虑采用并行流水线：语音识别和截图OCR已并行；也可让LLM生成答案的同时提前请求TTS流式合成，以缩短等待。

- **减少误触发和误识别**：为避免游戏音效触发麦克风错误识别，可实现**按键+语音**双确认或推拉开关模式，必要时提供PTT（Push-to-talk）开关按钮而非全程open mic。也可对识别结果置信度做判断，低置信度时提示用户确认或重复。
- **鲁棒性**：加强异常处理，比如：OCR没识别出文字时，依然可以返回通用回答而非报错；语音识别超时或失败时，退回文本输入模式提示用户。ADK代理服务掉线或出错时，在前端提示并尝试重启服务。
- **UI改进**: 美化浮窗界面风格，与游戏画面融合度更高（如半透明背景、游戏主题皮肤）。增加对话历史查看和复制功能，方便用户保存策略要点。语音方面，可提供音量控制和播放进度提示。
- **个性化**：让AI更具“陪伴感”，可在配置中添加“AI性格”选项，比如语气幽默风趣或严肃专业，由系统提示词控制Gemini的回答风格，满足不同玩家偏好。
- **多智能体扩展**：如果后续需要，ADK支持多智能体协作。可以考虑增加次要智能体处理特定任务（如一个Agent专管百科查询，一个专管聊天抚慰），主Agent与之通信后汇总答复。这属于扩展方向，在单人开发完成主要功能后再研究实现。

各阶段可以大约按上述顺序迭代，每阶段完成后进行测试验证，然后进入下一阶段开发。在功能逐步丰富的同时，保持代码稳定和可调试性。

关键技术要点与AI能力利用

在整个开发过程中，有几项关键的AI相关技术点需要特别关注和优化，以充分发挥Google Gemini及相关AI服务的能力：

中英双语支持

Gemini 2.5作为Google最先进的模型，具备强大的多语言能力，能自然地用中英文与用户交流。这一点将通过在提示中明确要求及样例引导来实现。例如系统指示模型：“根据用户使用的语言回答”，确保用户讲中文就用中文回复，讲英文就用英文。同时，Speech-to-Text和Text-to-Speech服务也需匹配语言：利用Google STT自动检测语言或根据游戏地区配置设定语言模型；TTS则挑选对应语言的合成声音。通过这些措施，GameFloaty的AI助理将能流畅地用用户母语交流，也能处理英文游戏术语，实现无缝的双语体验。

多模态交互

多模态能力指AI不仅能处理文本，还能利用视觉信息推理。Gemini据报道原生支持图像输入；若开放该功能，可直接馈入截图让模型解析。但在当前技术方案中，我们通过OCR作为替代，将视觉信息转成文本供模型使用。这仍然极大拓展了AI感知范围，让其“看到”游戏界面文字。借助ADK，我们可以进一步整合其他模态数据源，如游戏日志、玩家实时数据等，作为工具提供给AI。本规划以截图OCR为主，实现AI对游戏场景的感知。这使回答更具环境相关性，真正成为“贴在游戏画面上的智囊”。随着Google多模态模型能力开放，未来可升级为直接输入图像，更加高效精准。

上下文保持与多轮对话

上下文记忆是实现类似人类对话体验的关键。通过ADK的Session+Memory机制，GameFloaty会在内部维护用户与AI的对话历史。短期上下文（当前对话state）会每轮附加在提示中，让模型知道之前问了什么答了什么；长期有用的信息则可以存入Memory，在需要时调取。举例来说，用户先问了“Boss弱点是什么？”，AI回答后，如果下一问是“我该带什么武器？”，AI通过会话记忆知道这是关于同一个Boss的话题，会直接接续回答“建议使用水属性大剑…”。这种上下文关联在用户看来就是连续对话的效果。实现上要注意控制历史长度，Gemini虽强大但也有输入长度限制，ADK可帮助我们筛选哪些历史需要保留（例如重要的结论性语句）。我们也可以在每轮对话结束后，让模型自行总结这一轮的重要信息存入Memory（类似记笔记），以便对话很长时仍能回顾前情。通过精心设计，GameFloaty将能够进行长对话而不丢失主题，上下文理解大大增强。

总结：本开发计划通过模块化的分步实现，将在现有GameFloaty基础上引入强大的Gemini AI能力。利用Google ADK统一 orchestrate 语音、视觉和语言模型，实现游戏助手从简单百科查询工具升级为智能对话伙伴。整个方案兼顾技术可行性与产品体验：单人开发者可以按阶段完成，每阶段都有明确产出；最终用户将获得一个可语音交流、懂得查看游戏屏幕并提供针对性帮助的贴心“小浮窗”。通过最大化利用Gemini的多语言、多模态和推理优势，GameFloaty有望成为玩家游戏过程中真正“有温度、有趣味”的AI伙伴。各项功能在开发中若遇挑战，可充分参考Google提供的ADK文档和示例，借鉴最佳实践，不断迭代优化，最终达成产品目标。

To Jerry

https://google.github.io/adk-docs/

https://ai.google.dev/gemini-api/docs/live