<!-- Chosen Palette: Icy Blue & Warm Neutral -->

<!-- Application Structure Plan: 这不是一个网页应用，而是一个完整的 Windows 桌面系统架构。

1. 核心容器 **(Host)**: 使用 Unity 引擎 (C#) 作为3D房间、角色和UI的渲染与交互环境。
2. **OS集成层 (Bridge)**: 使用 C#/P-Invoke 调用原生 Windows API，实现将 Unity 窗口嵌入桌面层（壁纸层）。
3. 视觉追踪模块 **(Tracker)**: 使用 OpenCV (C# 封装) 访问摄像头，进行实时头部姿态估计。
4. 视差引擎 **(MagicWindow)**: Unity 摄像机模块，根据追踪数据实时修改“投影矩阵”(Projection Matrix)，实现真实视差。
5. **AI** 伙伴 **(Companion)**: 3D 角色模型 (VRM/Live2D in 3D)，由状态机（任务、闲聊、专注）驱动。
6. **AI** 大脑 **(Brain)**: C# 服务调用 Gemini API，并实现一个“工具调用(Function Calling)”注册表。
7. 本地工具 **(Tools)**: 一系列符合“工具规范”的 C# 模块（番茄钟、日历、文件系统），AI 大脑可以安全地调用它们。
8. 退出机制 (Egress): 全局热键与系统菜单，确保能随时安全退出并恢复原生桌面。
-->

<!-- Visualization & Content Choices:

1. 软件架构图: 目标=展示各模块关系 -> 方式=UML组件图 (HTML/CSS模拟) -> 交互=无 (静态图) -> 理由=清晰展示系统边界。
2. 头部追踪数据流: 目标=解释Magic Window原理 -> 方式=流程图 (HTML/CSS) -> 交互=无 -> 理由=说明从摄像头到3D相机的完整管线。
3. AI功能调用时序图: 目标=解释AI如何执行本地任务 -> 方式=UML时序图 (HTML/CSS) -> 交互=无 -> 理由=展示受控的Gemini API Function Calling流程。
-->

<!-- CONFIRMATION: NO SVG graphics used. NO Mermaid JS used. -->

《异世界窗户桌面系统》技术架构与实施蓝图 (v1.0)

基于您的需求规格 v1.0，我们制定以下技术架构方案，采用 **Unity (C#)** 作为核心开发引擎，辅以 **OpenCV** 进行视觉追踪，并集成 **Gemini API** 作为 AI 大脑。

1. 核心技术栈选型

- **3D** 引擎与主程序: **Unity 2022+ (C#)**
  - 理由: 强大的3D实时渲染能力、跨平台C#环境、成熟的资源商店（3D模型、动画工具）、相对轻量，且通过 P/Invoke (Platform Invoke) 能良好地调用原生 Windows API。
- 头部/眼动追踪: **OpenCV (Emgu CV C# 封装)**

    - 理由: 业界标准，提供稳定的人脸关键点检测与姿态估计算法（如 solvePnP），足以实现 Magic Window 需求。
- **AI 大脑: Google Gemini API**
    - 理由: 强大的 Function Calling 功能，是实现 AI 助手的核心，使其能安全调用本地受控工具。
- **OS 集成: C# P/Invoke (调用 user32.dll, dwmapi.dll)**
    - 理由: 实现 Windows 桌面层（壁纸层）嵌入的唯一可靠途径。

2. 整体软件架构

这是一个高层模块图，展示了系统的主要组件及其交互关系：

<h3 style="text-align: center; border-bottom: 2px solid #ddd; padding-bottom: 8px;">WWAW - 系统架构图</h3>

    用户 (输入)
    (头部/眼睛, 键盘/鼠标)

  <!-- 左侧：渲染与交互 -->

    核心：Unity 实时引擎 (C#)

        3D 房间与渲染
        (光照, 物理, 资源管理)

        AI 伙伴 (角色系统)
        (模型, 动画状态机, TTS)

```
        <div style="background: #eef; padding: 8px; border-radius: 4px;">
          <b>世界内 UI (功能模块)</b><br>
          (番茄钟, 日历, 记事本 - 3D表现)
        </div>

        <div style="background: #eef; padding: 8px; border-radius: 4px;">
          <b>视差引擎 (Magic Window)</b><br>
          (修改摄像机投影矩阵)
        </div>
      </div>
    </div>

    <!-- 右侧：服务与集成 -->
    <div style="border: 1px solid #ddd; padding: 10px; border-radius: 5px;">
      <b style="color: #c2185b;">服务与集成层 (C#)</b>
      <div style="margin-left: 15px; margin-top: 10px; display: grid; gap: 10px;">

        <div style="background: #ffebee; padding: 8px; border-radius: 4px;">
          <b>头部追踪模块 (OpenCV)</b><br>
          (摄像头 → 姿态数据)
        </div>

        <div style="background: #ffebee; padding: 8px; border-radius: 4px;">
          <b>AI 大脑 (Gemini API)</b><br>
          (对话, 决策, Function Calling)
        </div>

        <div style="background: #ffebee; padding: 8px; border-radius: 4px;">
          <b>受控工具层 (Tools)</b><br>
          (读/写文件, 读日历, 触发邮件)
        </div>

        <div style="background: #ffebee; padding: 8px; border-radius: 4px;">
          <b>OS 集成层 (P/Invoke)</b><br>
          (桌面嵌入, 全局热键, 退出)
        </div>
      </div>
    </div>
  </div>


</div>
```

3. 核心难点实现方案

3.1 难点一：Windows 桌面层集成 (模式 A)

需求: 应用必须作为桌面/壁纸层运行，而不是普通窗口。

方案: 我们将使用 P/Invoke 调用 user32.dll 来实现。

1. 启动: Unity 应用启动时，它首先是一个普通窗口。
2. 查找 **WorkerW**: Windows 11/10 的桌面图标层存在于一个名为 SHELLDLL_DefView 的窗口，它又是 Progman 的子窗口，而 Progman 又是 WorkerW 的子窗口。我们需要找到这个 WorkerW。
3. 发送消息: 我们向 Progman 发送一个特定的 Windows 消息 (0x052C)，这会强制系统在 Progman 后面生成一个新的 WorkerW 窗口（如果它不存在的话）。
4. 设置父窗口: 找到这个 WorkerW 窗口句柄（Handle）后，使用 SetParent API，将我们的 Unity 应用窗口句柄设置为这个 WorkerW 的子窗口。
5. 调整样式: 移除 Unity 窗口的边框和标题栏，使其填满 WorkerW。

这样，Unity 应用就“画”在了所有桌面图标的 后面，实现了完美的动态壁纸桌面。

退出机制: 必须注册一个全局热键（例如 Ctrl+Alt+Q），当触发时，Unity 应用调用 SetParent(self, NULL) 将自己释放回普通窗口，然后调用 Application.Quit() 正常退出。

3.2 难点二：Magic Window 视差实现

需求: 实时头部追踪，实现“魔法窗口”般的真实视差，而非简单平移。

方案: 关键在于修改摄像机的投影矩阵 **(Projection Matrix)**，而不是移动摄像机位置。这能创造出非对称视锥 (Asymmetric Frustum)，是实现真实视差（也称鱼缸VR）的标准方法。

数据管线:

1. 输入 **(OpenCV)**:
   - 通过摄像头捕获图像。
   - 使用人脸检测（Haar 特征或 DNN）找到面部。
   - 提取 68 个面部关键点。
   - 使用 solvePnP 算法，根据关键点和预定义的3D面部模型，估算出头部相对于摄像头的 6-DoF 姿态（位置 x,y,z 和旋转 pitch,yaw,roll）。
2. 平滑 **(Filter)**:
   - 原始数据会抖动。必须通过一个滤波器（如卡尔曼滤波器或简单的指数平滑）处理 x,y,z 值。
3. 映射 **(Unity)**:
   - 我们假定用户头部在现实中的坐标 (eye_x, eye_y, eye_z) 对应于虚拟世界中摄像机的眼睛位置。
   - “窗户”（即显示器）在虚拟世界中是一个固定的平面。
4. 计算投影矩阵 **(核心)**:

- Unity 的 Camera.main.projectionMatrix 可以被手动设置。
- 我们将根据 (eye_x, eye_y, eye_z) 和“窗户”平面的四个角，重新计算一个非对称的透视投影矩阵。
- 当用户向左移动（eye_x 变小），投影矩阵的中心点会向右偏移，导致渲染出的画面能看到更多右侧的内容，完全符合需求规格。

回退: 如果 OpenCV 未检测到人脸或摄像头未授权，此脚本自动禁用，projectionMatrix 恢复为 Unity 默认值，系统降级为“固定视角模式”。

3.3 难点三：AI 伙伴与本地工具调用 (Function Calling)

需求: AI 伙伴能通过 Gemini API 安全地访问本地权限（番茄钟、日历、文件）。

方案: 严格遵循 Gemini 的 Function Calling 机制，设计一个“受控工具层”。**AI** 永远不能直接执行系统命令。

交互时序:

1. 用户: （通过语音或文本）“帮我启动一个25分钟的番茄钟，任务是‘写报告’。”
2. **Unity (AI 大脑)**: 将此请求，连同一个“可用工具列表”发送给 Gemini API。
   - 可用工具列表 *(JSON Schema)*:
     - start_pomodoro(duration_minutes: int, task_name: string)
     - get_calendar_events(date: string)
     - list_files(directory: string)
3. **Gemini API (返回)**: API 不会直接回答。它会返回一个 tool_call 请求：
   - { "tool_call": { "name": "start_pomodoro", "arguments": { "duration_minutes": 25, "task_name": "写报告" } } }
4. **Unity (工具执行层)**:
   - AI 大脑收到此 tool_call。
   - 在“工具注册表”中查找名为 start_pomodoro 的本地 C# 函数。
   - 执行该函数：PomodoroManager.Instance.StartTimer(25, "写报告");
   - 此函数执行成功，并返回一个结果（例如 "{ "status": "success", "timer_id": 1 }" ）。
5. **Unity (AI 大脑)**: 将这个“工具执行结果”再次发送给 Gemini API。
6. **Gemini API (最终回复)**: 此时，API 知道工具已成功执行，它会生成一个自然的回复：
   - "好的，25分钟的‘写报告’番茄钟已经开始啦！要专注哦~"
7. **Unity (AI 伙伴)**:
   - 播放 TTS 语音：“好的，25分钟的‘写报告’番茄钟已经开始啦！要专注哦~”
   - AI 角色的动画状态机切换到 Focus 状态。
   - 3D 房间桌上的计时器 UI 开始倒计时。

这个架构完美地将 AI 决策（Gemini）与本地执行（C#）分离，确保了安全性和可控性，完全满足您的需求。

4. 下一步开发计划

这个蓝图是我们实现项目的基石。根据这个架构，我们可以开始逐个模块地生成代码。

我建议的第一个开发冲刺 (Sprint 1) 目标是：实现 **Magic Window** 核心体验。

我们可以从以下两个关键脚本开始：

1. **HeadTracker.cs**: 使用 Emgu CV (OpenCV) 捕获摄像头并估算头部 x,y,z 坐标。
2. **MagicWindowCamera.cs**: 获取 HeadTracker 的坐标，并实时修改 Camera.main.projectionMatrix。

您准备好开始生成第一个核心模块的代码了吗？