✅ UnityPilot – System Design Document (v1.0)

📘 1. Overview

Name: UnityPilot
Type: Unity Editor Extension
Goal: Provide AI-powered assistance to Unity developers inside the Unity Editor to improve speed, quality, and structure of game development.

Scope:

- Support natural language prompts inside the Unity Editor
- Generate Unity scripts, prefab scaffolds, and game logic based on developer requests
- Perform project audits for common pitfalls
- Enable guided game assembly (like Tower Defense) using existing assets

---

👤 2. Target Users

- Indie game developers and solo Unity creators
- Small to mid-size teams needing productivity tooling
- Educators/students looking to learn Unity with AI support

🎯 3. Goals & Phases

🛠️ MVP Goals

- Prompt-based script generation
- Prompt-based prefab/GameObject creation (using prefab registry)
- Missing script / broken reference detection
- Configurable AI model support (OpenAI, Claude, Gemini via adapters)
- Basic prompt templates (e.g., “Create a MonoBehaviour that spawns bullets”)

🚀 Post-MVP Goals

- Guided scene/game logic assembly (e.g., Tower Defense builder)
- Design pattern assistants (e.g., Singleton, MVC, ScriptableObject patterns)
- Asset usage audits and recommendations
- Refactor/explain code tools
- Unity test stub generation
- Plug-in API for alternate AI providers

🧱 4. System Architecture

UnityPilot is designed as a modular Unity Editor extension with clearly separated responsibilities, allowing for easy extensibility and maintainability.

🔹 Component Overview

Component	Responsibilities
Unity Editor Window	- Main UI for prompts, responses, and templates- Trigger actions like audits
Prompt Handler	- Processes raw user input- Selects or applies prompt templates- Classifies task type

AI Adapter Layer	- Interfaces with AI APIs- Supports multiple providers via plugin-style adapters
Code Generator & Parser	- Validates AI responses- Writes code to file system- Adds `TODOs` or logs errors
Scene & Project Manager	- Places prefabs or components in scenes- Connects logic to generated scripts- Runs project audits

---

🔹 Data Flow Diagram (Markdown-style)

[Unity Editor Window]
↓
[Prompt Handler]
↓
[AI Adapter Layer]
↓
[Code Generator & Parser]
↓
[Scene & Project Manager]

Each component is loosely coupled and communicates in one direction, allowing you to replace or extend functionality (e.g., swap OpenAI with Claude).

🔹 Adapter Pattern Example

```
public interface IAIProvider {
    Task<string> GenerateResponse(string prompt);
}
```

Implementations like `OpenAIAdapter`, `ClaudeAdapter`, etc., conform to this interface and can be injected at runtime via settings or config.

🔹 Scene & Prefab Generation Pipeline

Step	Responsibility
Parse Prompt	PromptHandler
Query AI	AIAdapterLayer
Validate/Format Code	CodeGenerator
Instantiate Prefabs	SceneManager
Hook scripts/components	SceneManager
Report result to user	UnityEditorWindow

🧩 5. Component Modules

5.1 Unity Editor Window

- UI/UX dockable editor panel
- Prompt input field, output display
- Dropdown for prompt templates
- Button for “Generate” or “Run Audit”

5.2 Prompt Engine

- Prompt templates (configurable via JSON)
- Prompt expansion (e.g., insert script name, variable)
- Prompt sanitization and classification (e.g., “codegen”, “scene-gen”, “audit”)

5.3 AI Adapter Layer

```
public interface IAIProvider {
    Task<string> GenerateResponse(string prompt);
}
```

- Implementations:
    - OpenAIAdapter
    - ClaudeAdapter
    - GeminiAdapter
- API key or token loaded via secure config

5.4 Code Generator

- Receives raw AI output (C# code)
- Validates code (basic syntax checks)
- Writes script files to correct Unity folder
- Optionally opens file in external editor

5.5 Scene & Prefab Generator

- Uses existing prefab registry
- Places prefabs in scene via script
- Attaches AI-generated components
- Sets positions, hierarchy (limited, rule-based)

5.6 Project Auditor

- Detects:
    - Missing scripts
    - Broken references

    - Components with errors
- Future: unused assets, large textures, optimization flags

🧠 6. Prompt Scenarios (Examples)

Codegen Prompt

> “Create a MonoBehaviour that rotates an object at 60 degrees per second”

Guided Scene Prompt

> “Add a turret from turret.prefab at (0, 0, 0) and make it shoot enemies using TurretAI.cs”

Audit Prompt

> “Check my scene for broken scripts and report issues”

🧰 7. Technologies

Area	Tool/Framework
Unity	Unity Editor (2021+)
Language	C#
AI API	OpenAI / Claude (via HTTP)
Config Format	JSON / ScriptableObjects
File I/O	UnityEditor.AssetDatabase, System.IO
Code Validation	Regex/syntax checks (initial), later Roslyn (optional)

🔌 8. Extensibility Plan

- **Prompt Plugin Registry**: Easily register new prompt templates
- **AI Adapter Interface**: Add any API-based AI with minimal effort
- **Scene Action Handlers**: Plugin system for custom prefab actions (e.g., wire spawner to enemies)

📅 9. Roadmap

MVP Milestones

- Prompt window UI + prompt processor
- Script generation with OpenAI
- Script file creation & injection
- Basic audit system
- Prefab registry + placement handler
- AI Adapter interface

Post-MVP

- Tower Defense Builder (guided copilot mode)
- Advanced audits (asset analysis)
- Refactor/explain tools
- Prompt versioning (opt-in)
- Contributor setup + open source

📎 10. Risks & Constraints

Risk	Mitigation
AI hallucination or broken code	Add TODOs, comments, validate code
API cost or rate limits	Allow BYO API key
Complex scene layout generation	Use structured templates + prefab registry
Users misunderstanding AI output	Add explanation + warnings

✅ Summary

UnityPilot will be:

- ⚡ Fast to use
- 🧠 AI-enhanced but safe
- 🛠️ Focused on productivity, not hype
- 🤝 Built for real Unity workflows
