Small Space Optimization Agent

One-Line Summary

An agentic spatial planner that optimizes small rooms for movement and multi-use, while allowing users to lock key furniture and letting the agent intelligently re-optimize the rest of the space.

What We Are Building

A **human-in-the-loop spatial intelligence system** that:

- Understands a real room from a photo
- Finds an optimal baseline layout
- Lets the user fix one object (bed or desk)
- Automatically re-plans all remaining objects
- Preserves realism through targeted image edits
- Explains why each change was made

Gemini is used for vision and image editing, while planning, constraints, and iteration live outside the model.

Problem Statement

The Core Problem

Small rooms fail due to **poor spatial decisions**, not aesthetics.

Users struggle with:

- Blocked walking paths

- Doors colliding with furniture
- No separation between work and rest
- One fixed furniture decision breaking the entire layout

Existing tools:

- Either fully auto-generate designs with no user control
- Or rely on manual drag-and-drop with no intelligence

There is no system that **collaborates with the user while enforcing spatial logic**.

Pain Points We Are Targeting

1. **Lack of spatial reasoning**
   Current tools do not reason about movement, clearance, or flow.
2. **No partial control**
   Users often know one thing they want fixed, but tools cannot adapt around it.
3. **Trial-and-error layouts**
   One manual change forces users to redesign everything.
4. **Unrealistic AI outputs**
   Generated rooms ignore physical constraints and real usability.

Our Solution

We solve this by combining:

- Agent-led planning

- User-imposed constraints (object locking)
- Continuous re-optimization

The agent adapts to the user, not the other way around.

Technical Architecture

High-Level Flow

```
User Photo
  ↓
Vision & Reasoning (Gemini)
  ↓
Structured Room Graph (Stateful)
  ↓
Constraint Engine
  ↓
Layout Planner (Iterative)
  ↓
User Locks One Object
  ↓
Re-Optimization Loop
  ↓
Targeted Image Editing (Gemini)
  ↓
Explained Visual Outputs
```

1. Vision and Object Extraction (Gemini)

Input:

- Single photo of a small bedroom

Output:

- Objects detected (bed, desk, chair, door, window)
- Approximate bounding boxes or relative positions
- Classification: fixed or movable

This output is structured JSON, not free text.

2. Structured Room Graph (Outside Gemini)

We maintain a lightweight internal model:

- Nodes: furniture and structural elements
- Attributes:
  - Position
  - Size estimate
  - Locked or unlocked state

This graph persists across iterations.

3. Constraint Engine (Deterministic)

Hard constraints:

- Door must remain unblocked
- Minimum walking clearance
- Locked objects cannot move

Soft constraints:

- Desk near window

- Bed away from door

Constraints are enforced outside Gemini.

4. Layout Planner (Agent Loop)

The planner:

1. Proposes a layout
2. Validates constraints
3. Scores usability
4. Accepts or revises

When a user locks an object, the planner restarts with updated constraints.

5. Targeted Image Editing (Gemini)

Gemini is used only to:

- Move or replace unlocked objects
- Preserve lighting and perspective
- Avoid regenerating the entire image

Edits are surgical, not global.

6. Explainability Layer

Each output includes:

- Highlighted locked object
- Walking path overlay
- Short explanation of tradeoffs

This is critical for judge clarity.

Development Plan (Parallel Execution)

Assumption:

- 3 developers
- ~14 days
- Everyone works simultaneously

Developer A (Vision + Image Editing)

Days 1 to 4

- Gemini prompts for object detection
- JSON schema for room extraction
- Initial image edit prompts

Days 5 to 9

- Targeted object movement prompts
- Preserve lighting and geometry
- Handle locked vs unlocked objects

Days 10 to 14

- Refine realism
- Handle failure cases
- Support demo scenarios

Developer B (Core Intelligence)

Days 1 to 4

- Room graph data structure
- Object locking logic
- State persistence

Days 5 to 9

- Constraint engine
- Clearance heuristics
- Layout scoring

Days 10 to 14

- Iterative planner loop
- Re-optimization logic
- Integration testing

Developer C (Frontend + Demo)

Days 1 to 4

- Image upload flow
- Basic UI scaffolding

Days 5 to 9

- Lock object selection
- Before and after comparison
- Overlay visualization

Days 10 to 14

- Explainability UI
- Demo polish
- Video and submission assets

Final MVP Scope (Locked)

We will support:

- One room type (small bedroom)
- One locked object at a time
- One main constraint (walking clearance)
- Two to three optimized layouts

This is enough to demonstrate:

- Agentic reasoning
- Human-in-the-loop interaction

- Clear differentiation from Gemini alone

Why This Wins at Gemini Hackathon

- Uses Gemini meaningfully, not superficially
- Demonstrates long-horizon reasoning
- Shows planning, iteration, and adaptation
- Solves a real, relatable problem
- Easy for judges to understand in under 30 seconds

Tech Stack

- **Core AI & Vision:**
  - **Google Vertex AI:** Primary infrastructure for accessing Gemini 1.5 Pro (Vision) and Flash (Reasoning).
  - **google-genai:** Unified SDK for Python.
  - **Instructor:** Middleware to force deterministic, structured JSON output from Gemini.
- **Agentic Logic & Spatial Reasoning:**
  - **LangGraph:** Orchestrates the stateful "Human-in-the-loop" workflow and re-optimization cycles.
  - **Shapely:** Computational geometry engine for defining furniture polygons and calculating collisions/clearance.
  - **NetworkX:** Manages the "Room Graph" to track relationships between objects (e.g., "nightstand" must be near "bed").
- **Backend & API:**
  - **FastAPI:** High-performance Python web server to host the agent logic.
  - **Pydantic:** Data validation and schema definition for IO.
- **Frontend:**

- **Next.js (React):** Framework for the user interface.
- **React-Konva:** Canvas library for drawing overlays and handling interactive "object locking" on top of the room image.

Data Schemas

1. Vision Output (Gemini -> Python): The Vision Agent must return this exact structure:

JSON
```
{
  "room_dimensions": {"width_estimate": 300, "height_estimate": 400},
  "objects": [
    {
      "id": "bed_1",
      "label": "bed",
      "bbox": [10, 10, 100, 200], // [x, y, width, height]
      "type": "movable", // or "structural"
      "orientation": 0 // degrees
    }
  ]
}
```

2. The Graph State (LangGraph): The shared state passed between nodes in the agent loop:

Python
```
class AgentState(TypedDict):
    image_base64: str
    current_layout: List[RoomObject]
    locked_object_ids: List[str]
    constraint_violations: List[str]
    iteration_count: int
```

2. The API Interface (Frontend <-> Backend)

API Endpoints (FastAPI)

- `POST /analyze`: Accepts raw image -> returns parsed JSON layout.
- `POST /optimize`: Accepts `{ current_layout, locked_ids }` -> returns `{ new_layout, explanation }`.
- `POST /render`: Accepts `{ final_layout }` -> returns `{ image_url }` (using Gemini Inpainting).

3. The LangGraph Node Structure

LangGraph Workflow Definition The graph should consist of these distinct nodes:

1. `VisionNode`: Calls Vertex AI Gemini 1.5 Pro to extract JSON.
2. `ConstraintNode`: Uses Shapely to check overlaps and path widths.
3. `SolverNode`: Heuristic logic to move unlocked items to valid free space.
4. `ReviewNode`: (Optional) Gemini Flash checks if the layout "makes sense" human-wise.
5. `RenderNode`: Calls Gemini 1.5 Pro to generate the final image edit.

4. The Project Structure (Scaffold)

Desired Folder Structure

Plaintext

```
/backend
  /app
    /agents (LangGraph nodes)
    /core (Shapely logic & geometry utils)
    /models (Pydantic schemas)
    main.py (FastAPI entrypoint)
/frontend
  /components (React-Konva canvas)
  /hooks (API calls)
```