# Project Blueprint: The Heist Architect Framework

## 1. Concept & Objective

The project creates a dual-agent system where two intelligences compete in a "Security vs. Infiltration" loop.

+1

* **The Architect:** A generative agent that designs security layouts (walls, cameras, guards) to detect intruders.  
  +2
* **The Solver:** A navigation agent that attempts to reach a vault while remaining undetected.  
  +1
* **Core Research Theme:** Investigating how adversarial competition leads to the emergence of sophisticated spatial and temporal strategies, such as timing camera rotations and exploiting patrol blind spots.  
  +1

## 2. Environment Architecture

The environment is a 2D grid-based simulation that functions as the "battleground" for both agents.

A. Spatial Components

+3

* **Grid Layout:** A defined area (e.g., 20x20 or 50x50 tiles).
* **Obstacles:** Static walls and corridors that block movement and vision.  
  +1
* **The Vault:** The target objective for the Solver.  
  +1

B. Dynamic Security Components

* **Rotating Cameras:** Fixed units with a specific Field of View (FOV) and rotation speed.  
  +1
* **Patrol Guards:** Moving units with defined paths that the Architect must designate.  
  +1
* **Visibility Map:** A time-dependent layer showing which tiles are currently under surveillance.

## 3. Agent Definitions & Action Spaces

| **Agent** | **Roles & Responsibilities** | **Action Space (What it can do)** |
| --- | --- | --- |
| **Architect** | Designs the challenge | Place walls, cameras (speed/FOV), and guards/patrols |
| **Solver** | Becomes the "Spy" | Move in continuous time, hide behind walls, and wait for openings |

## 4. State Representation (Inputs)

To learn, each agent must "see" the environment in a specific way:

* **Occupancy Grid:** A map showing where walls and objects are located.
* **Dynamic Visibility Map:** A live feed of which areas are "red zones" (visible to cameras/guards) at any given moment.
* **Relative Coordinates:** The Solver's distance to the vault and its own visibility footprint.

## 5. The Game-Theoretic Reward System

Rewards are designed to create a "zero-sum" competition where one agent's loss is the other's gain.

For the Architect

* **Positive Reward (+1):** Every time the Solver is detected.
* **Negative Penalty (-1):** If the generated level is physically impossible to solve (no path to the vault exists).

For the Solver

* **Positive Reward (+10):** Successfully reaching the vault.
* **Negative Penalty (-1):** Getting detected by a camera or guard.

## 6. Implementation Phases (How to Build It)

### Phase 1: The Static Environment (Week 1-2)

* Build the 2D grid and the "Validity Checker" (ensures there is always a path from start to vault).
* Implement basic vision cones for cameras and guards using Raycasting.

### Phase 2: The Solver Training (Week 3-4)

* Train the Solver on **fixed, human-designed levels** first.
* The Solver must learn the basics of "staying out of the red cones."

### Phase 3: The Adversarial Loop (Week 5-8)

* Introduce the Architect. Initially, give the Architect a small budget (e.g., 1 camera, 2 walls).
* **Training Loop:**
  1. Architect generates a level.
  2. Solver attempts the level 100 times.
  3. Rewards are distributed to both based on success/detection rates.
  4. Both agents update their neural networks.

### Phase 4: Scaling & Complexity (Week 9+)

* Increase the Architect's budget (more guards, faster cameras).
* Introduce **temporal complexity**, where guards have irregular patrol patterns.

7. Scientific Novelty

Unlike traditional Reinforcement Learning where the environment is static, your framework features **Non-Stationary Adversarial Learning**. This means the difficulty is **emergent**—it grows naturally as the agents get smarter, mimicking a real-world security evolution.

+1

**Next Step:** Would you like me to draft a **Budget Table** that defines the "costs" for the Architect's items (e.g., how much a guard costs vs. a camera) to ensure the game remains balanced?