Turitto

Giuseppe Turitto <giuseppe@turitto.com>

Docs

Giuseppe Turitto <giuseppe.turitto@kiwi.com> Sat, Oct 4, 2025 at 6:28 PM
To: giuseppe@turitto.com

Here is the detailed **Product Requirements Document (PRD)** for **Labyrinth: The Temporal Hunt**, which serves as the single source of truth for the game's scope and functional requirements.

Product Requirements Document (PRD v1.0)

Labyrinth: The Temporal Hunt

Field	Value
Product Goal	To create a turn-based, multi-agent text survival horror simulator that demonstrates complex AI strategy and deterministic game logic using a clean JSON-API interface.
Target User	AI Agent (Mistral), human player proxy, and future visual front-ends.
Current Phase	**Minimum Viable Product (MVP)** – Fully functional text-based simulation with learning integration.
Author	Gemini
Status	Accepted

---

1. Game Environment and Physics (The GSM Domain)

The **Game State Manager (GSM)** is responsible for all deterministic physics and map rules.

1.1 Spatial Simulation

ID	Requirement	Specification
F.R. 1.1.1	**Scale & Bounds**	The simulation space must be a conceptual **kilometer cube**.
F.R. 1.1.2	**Grid Unit**	All location tracking and movement calculations must use a discrete grid unit of **meters**.
F.R. 1.1.3	**Start Position**	**User:** Center of the cube . **Minotaur:** At the perimeter of the level (e.g.,).
F.R. 1.1.4	**Verticality**	Levels (changes in Z-coordinate) must be unique maze layouts, connected only by **Ramps**.

ID	Requirement	Specification
F.R. 1.1.5	**Collision Logic**	User movement requests (steps=100) must halt immediately upon hitting a collision (wall, obstacle, map edge), returning the actual steps_moved . **Squeezing past the Minotaur is impossible.**

1.2 Movement and Time

ID	Requirement	Specification
F.R. 1.2.1	**Walk/Crawl**	**Speed 1:** 1 step/second (Low Noise, Stamina Recovery).
F.R. 1.2.2	**Run**	**Speed 2:** 2 steps/second (High Noise, Stamina Depletion). This is the maximum movement rate.
F.R. 1.2.3	**Time Calculation**	The GSM must accurately return the time_taken metric, calculated as . All timers (Cooldowns, Paralysis) must advance by this time_taken value.
F.R. 1.2.4	**Stamina**	Stamina must be tracked (0-100%). Running drains stamina; walking/halting allows slow recovery. Zero stamina limits speed to **1 step/sec**.

1.3 Sensory and Visibility

ID	Requirement	Specification
F.R. 1.3.1	**User Sight**	The User can only perceive items/paths within a **2-step (5 meter)** radius.
F.R. 1.3.2	**Minotaur Sight**	The Minotaur must be simulated with an **Infrared sight** radius of **6 steps (15 meters)**, giving it a crucial LOS advantage.
F.R. 1.3.3	**Audio Cues**	The GSM must categorize Minotaur distance into NONE , FAINT , CLOSE , VERY CLOSE based on proximity, derived from physical distance in .

---

2. Minotaur Temporal System and AI

The Minotaur's unique threat stems from its control over the fourth dimension (Time).

2.1 Temporal Jump Logic

ID	Requirement	Specification
F.R. 2.1.1	**Positional Jump**	The jump must be **positional only**. The Minotaur vanishes at and reappears at the **exact same** location after the duration.
F.R. 2.1.2	**Duration**	The vanish/re-entry cycle must consume a randomized duration of **5 to 10 seconds** of game time.

ID	Requirement	Specification
F.R. 2.1.3	**Jump Cooldown**	The ability must have a global cooldown of **600 seconds (10 minutes)**, tracked by the GSM.
F.R. 2.1.4	**Cue Output**	The JSON must include the `temporal_status` (`VANISHING`, `VANISHED`, `RE_ENTRY_ALERT`).

2.2 Lantern and Paralysis

ID	Requirement	Specification
F.R. 2.2.1	**Paralysis Effect**	When the user executes `USE LANTERN`, the Minotaur's status must transition to **`PARALYZED`** for **120 seconds (2 minutes)**.
F.R. 2.2.2	**Lantern Uniqueness**	Only one **LANTERN** item can exist in the game environment at any time.
F.R. 2.2.3	**Respawn Cooldown**	After the Lantern is used, the system must wait **12 minutes (720 seconds)** before a new Lantern is eligible to spawn randomly.

---

3. Agent and Communication Protocol

The architecture must support the communication required for the multi-agent system.

3.1 Data Integrity

ID	Requirement	Specification
F.R. 3.1.1	**User Command**	User inputs must be validated against the **`UserInput`** **Pydantic schema**.
F.R. 3.1.2	**Minotaur Decision**	The Minotaur Agent's output must be validated against the **`MinotaurDecision`** **Pydantic schema**.
F.R. 3.1.3	**System Output**	The GSM's final output to the front-end must strictly conform to the **`GameStateResponse`** **JSON schema**.

3.2 Iterative Learning (NFR)

ID	Requirement	Specification
N.F.R. 3.2.1	**Memory Persistence**	The `Memory Manager` must persistently store structured reflections (`KeyReflection`, `Outcome`) from past runs to a local file (`experience_log.json`).
N.F.R. 3.2.2	**Reflection Injection**	At the start of a new game, the system must retrieve relevant reflections and inject them into the **System Prompt** of both the Minotaur and User Agents to inform their strategy.

3.3 Objectives and End Conditions

ID	Requirement	Specification
F.R. 3.3.1	**Escape Condition**	The system must transition the status to `ESCAPED` when the User's inventory contains the **Red, Blue, and Yellow Stones**.
F.R. 3.3.2	**Loss Condition**	The system must transition the status to `DEATH` immediately upon the Minotaur and User being in the **same grid unit** (`x, y, z`) while the Minotaur is in `CHASING_3D` status.

Here is the detailed **Product Requirements Document (PRD)** for **Labyrinth: The Temporal Hunt**, which serves as the single source of truth for the game's scope and functional requirements.

Product Requirements Document (PRD v1.0)

Labyrinth: The Temporal Hunt

Field	Value
Product Goal	To create a turn-based, multi-agent text survival horror simulator that demonstrates complex AI strategy and deterministic game logic using a clean JSON-API interface.
Target User	AI Agent (Mistral), human player proxy, and future visual front-ends.
Current Phase	**Minimum Viable Product (MVP)** – Fully functional text-based simulation with learning integration.
Author	Gemini
Status	Accepted

---

1. Game Environment and Physics (The GSM Domain)

The **Game State Manager (GSM)** is responsible for all deterministic physics and map rules.

1.1 Spatial Simulation

ID	Requirement	Specification
F.R. 1.1.1	**Scale & Bounds**	The simulation space must be a conceptual **kilometer cube**.
F.R. 1.1.2	**Grid Unit**	All location tracking and movement calculations must use a discrete grid unit of **meters**.
F.R. 1.1.3	**Start Position**	**User:** Center of the cube . **Minotaur:** At the perimeter of the level (e.g.,).

ID	Requirement	Specification
F.R. 1.1.4	**Verticality**	Levels (changes in Z-coordinate) must be unique maze layouts, connected only by **Ramps**.
F.R. 1.1.5	**Collision Logic**	User movement requests (steps=100) must halt immediately upon hitting a collision (wall, obstacle, map edge), returning the actual steps_moved . **Squeezing past the Minotaur is impossible.**

1.2 Movement and Time

ID	Requirement	Specification
F.R. 1.2.1	**Walk/Crawl**	**Speed 1:** 1 step/second (Low Noise, Stamina Recovery).
F.R. 1.2.2	**Run**	**Speed 2:** 2 steps/second (High Noise, Stamina Depletion). This is the maximum movement rate.
F.R. 1.2.3	**Time Calculation**	The GSM must accurately return the time_taken metric, calculated as . All timers (Cooldowns, Paralysis) must advance by this time_taken value.
F.R. 1.2.4	**Stamina**	Stamina must be tracked (0-100%). Running drains stamina; walking/halting allows slow recovery. Zero stamina limits speed to **1 step/sec**.

1.3 Sensory and Visibility

ID	Requirement	Specification
F.R. 1.3.1	**User Sight**	The User can only perceive items/paths within a **2-step (5 meter)** radius.
F.R. 1.3.2	**Minotaur Sight**	The Minotaur must be simulated with an **Infrared sight** radius of **6 steps (15 meters)**, giving it a crucial LOS advantage.
F.R. 1.3.3	**Audio Cues**	The GSM must categorize Minotaur distance into NONE , FAINT , CLOSE , VERY CLOSE based on proximity, derived from physical distance in .

---

2. Minotaur Temporal System and AI

The Minotaur's unique threat stems from its control over the fourth dimension (Time).

2.1 Temporal Jump Logic

ID	Requirement	Specification
F.R. 2.1.1	**Positional Jump**	The jump must be **positional only**. The Minotaur vanishes at and reappears at the **exact same** location after the duration.

ID	Requirement	Specification
F.R. 2.1.2	**Duration**	The vanish/re-entry cycle must consume a randomized duration of **5 to 10 seconds** of game time.
F.R. 2.1.3	**Jump Cooldown**	The ability must have a global cooldown of **600 seconds (10 minutes)**, tracked by the GSM.
F.R. 2.1.4	**Cue Output**	The JSON must include the `temporal_status` (`VANISHING`, `VANISHED`, `RE_ENTRY_ALERT`).

2.2 Lantern and Paralysis

ID	Requirement	Specification
F.R. 2.2.1	**Paralysis Effect**	When the user executes `USE LANTERN`, the Minotaur's status must transition to **`PARALYZED`** for **120 seconds (2 minutes)**.
F.R. 2.2.2	**Lantern Uniqueness**	Only one **LANTERN** item can exist in the game environment at any time.
F.R. 2.2.3	**Respawn Cooldown**	After the Lantern is used, the system must wait **12 minutes (720 seconds)** before a new Lantern is eligible to spawn randomly.

---

3. Agent and Communication Protocol

The architecture must support the communication required for the multi-agent system.

3.1 Data Integrity

ID	Requirement	Specification
F.R. 3.1.1	**User Command**	User inputs must be validated against the **`UserInput`** **Pydantic schema**.
F.R. 3.1.2	**Minotaur Decision**	The Minotaur Agent's output must be validated against the **`MinotaurDecision`** **Pydantic schema**.
F.R. 3.1.3	**System Output**	The GSM's final output to the front-end must strictly conform to the **`GameStateResponse`** **JSON schema**.

3.2 Iterative Learning (NFR)

ID	Requirement	Specification
N.F.R. 3.2.1	**Memory Persistence**	The `Memory Manager` must persistently store structured reflections (`KeyReflection`, `Outcome`) from past runs to a local file (`experience_log.json`).

ID	Requirement	Specification
N.F.R. 3.2.2	**Reflection Injection**	At the start of a new game, the system must retrieve relevant reflections and inject them into the **System Prompt** of both the Minotaur and User Agents to inform their strategy.

3.3 Objectives and End Conditions

ID	Requirement	Specification
F.R. 3.3.1	**Escape Condition**	The system must transition the status to `ESCAPED` when the User's inventory contains the **Red, Blue, and Yellow Stones**.
F.R. 3.3.2	**Loss Condition**	The system must transition the status to `DEATH` immediately upon the Minotaur and User being in the **same grid unit** (x, y, z) while the Minotaur is in **`CHASING_3D`** status.

Architectural Decision Records (ADRs)

These documents formally record the significant architectural decisions made for the **Labyrinth: The Temporal Hunt** project, ensuring clarity, context, and traceability.

ADR 001: Agent Orchestration and Control Framework

Field	Value
Title	Selection of LangGraph for Stateful, Deterministic Agent Orchestration
Status	Accepted
Date	October 4, 2025

Context

The core challenge is orchestrating the turn-based game loop, which involves deterministic physics (managed by the **Game State Manager, or GSM**) and cognitive decision-making (managed by the **LLMs**). The sequence of actions is not free-form conversation but a rigid state flow where logic relies heavily on timers, cooldowns, and conditional branching (e.g., if Minotaur Cooldown is zero, jump is possible). Generic conversational frameworks often fail to provide this necessary level of control and state persistence.

Decision

We will use **LangGraph** to model the multi-agent workflow. LangGraph allows us to define the game's execution path as a **Directed Acyclic Graph (DAG)**, treating the GSM and the LLM calls as distinct, ordered **nodes**. This framework enforces a controlled state machine: User Action GSM Update Minotaur Decision GSM Update.

Consequences

Positive	Negative
High Control: Provides explicit control over the turn order and conditional branching based on deterministic metrics (e.g., collision results, timer values).	**Steeper Learning Curve:** Requires the team to understand graph theory and state management within LangGraph, increasing initial development complexity.
Stateful Management: LangGraph natively supports managing a **shared, central state** object, making it ideal for tracking mutable game metrics like **User Position**, **Minotaur Cooldown**, and **Inventory**.	**Framework Lock-in:** The core application logic becomes deeply dependent on the LangGraph framework structure.
Tool Integration: Natively supports integrating the GSM's deterministic functions (like calculating a collision or running the jump timer) as tools that the LLMs can call, maintaining the logic/cognition separation.	

ADR 002: Selection of Large Language Models (LLMs)

Field	Value
Title	Assignment of Gemini to Minotaur and Mistral to User Agent Roles
Status	Accepted
Date	October 4, 2025

Context

To maximize the asymmetrical horror and demonstrate distinct cognitive strategies, two powerful LLMs are necessary. The Minotaur requires abstract reasoning over non-linear concepts (like time), while the User Agent requires highly efficient, risk-averse tactical planning.

Decision

1. **Minotaur Agent:** Will be powered by **Gemini** (via `google-genai` SDK).
2. **User Agent:** Will be powered by **Mistral** (via `mistralai` SDK).

Consequences

Positive	Negative
Cognitive Alignment: Gemini excels at complex, multi-step reasoning, aligning with the Minotaur's need to understand and exploit temporal dynamics and sound signatures.	**API Management Overhead:** Requires managing and monitoring the usage, billing, and credentials for **two separate LLM providers**.
Performance Efficiency: Mistral is known for high efficiency and speed when handling structured, tactical reasoning tasks, making it a cost-effective choice for the high volume of tactical decisions the User Agent makes.	**Prompt Sensitivity:** Extensive testing and prompt engineering will be required to ensure both models consistently output the exact JSON structure defined by the **Pydantic schemas** (see ADR 003).
Learning Richness: The strategic contrast between the two models provides richer failure data for the **Memory**	

Positive	Negative
Manager to analyze and feed back into future game runs.	

ADR 003: Communication Protocol

Field	Value
Title	Enforcement of JSON Communication via Pydantic
Status	Accepted
Date	October 4, 2025

Context

In LLM-driven games, the primary source of instability is the LLM hallucinating output formats or ignoring constraints. For this project, every action (movement, jump, grab) must be passed to and from the deterministic GSM via a reliable, parseable interface.

Decision

All inputs (`UserInput` , `MinotaurDecision`) and outputs (`GameStateResponse`) will be defined using **Pydantic models**. LangGraph will be configured to use Pydantic schemas in its tool-calling mechanism, forcing the LLMs to output valid JSON before the GSM processes the command.

Consequences

Positive	Negative
Data Integrity: Eliminates LLM hallucination from breaking the core game loop, as malformed output is caught by the Pydantic validator before reaching the GSM logic.	**Increased Development Overhead:** Requires writing verbose Pydantic schemas for every structured piece of data, adding initial boilerplate code.
Scalability to Visualization: The structured JSON output is immediately compatible with any modern front-end framework (Streamlit, React, Vue), facilitating the future **Visual MVP** phase.	
Traceability: Every turn is a structured JSON record, making debugging, logging, and iterative learning straightforward and reliable.	# Architectural Decision Records (ADRs) These documents formally record the significant architectural decisions made for the **Labyrinth: The Temporal Hunt** project, ensuring clarity, context, and traceability.

Positive	Negative

ADR 001: Agent Orchestration and Control Framework

Field	Value
Title	Selection of LangGraph for Stateful, Deterministic Agent Orchestration
Status	Accepted
Date	October 4, 2025

Context

The core challenge is orchestrating the turn-based game loop, which involves deterministic physics (managed by the **Game State Manager, or GSM**) and cognitive decision-making (managed by the **LLMs**). The sequence of actions is not free-form conversation but a rigid state flow where logic relies heavily on timers, cooldowns, and conditional branching (e.g., if Minotaur Cooldown is zero, jump is possible). Generic conversational frameworks often fail to provide this necessary level of control and state persistence.

Decision

We will use **LangGraph** to model the multi-agent workflow. LangGraph allows us to define the game's execution path as a **Directed Acyclic Graph (DAG)**, treating the GSM and the LLM calls as distinct, ordered **nodes**. This framework enforces a controlled state machine: User Action GSM Update Minotaur Decision GSM Update.

Consequences

Positive	Negative
High Control: Provides explicit control over the turn order and conditional branching based on deterministic metrics (e.g., collision results, timer values).	**Steeper Learning Curve:** Requires the team to understand graph theory and state management within LangGraph, increasing initial development complexity.
Stateful Management: LangGraph natively supports managing a **shared, central state** object, making it ideal for tracking mutable game metrics like **User Position**, **Minotaur Cooldown**, and **Inventory**.	**Framework Lock-in:** The core application logic becomes deeply dependent on the LangGraph framework structure.
Tool Integration: Natively supports integrating the GSM's deterministic functions (like calculating a collision or running the jump timer) as tools that the LLMs can call, maintaining the logic/cognition separation.	

Positive	Negative

ADR 002: Selection of Large Language Models (LLMs)

Field	Value
Title	Assignment of Gemini to Minotaur and Mistral to User Agent Roles
Status	Accepted
Date	October 4, 2025

Context

To maximize the asymmetrical horror and demonstrate distinct cognitive strategies, two powerful LLMs are necessary. The Minotaur requires abstract reasoning over non-linear concepts (like time), while the User Agent requires highly efficient, risk-averse tactical planning.

Decision

1. **Minotaur Agent:** Will be powered by **Gemini** (via `google-genai` SDK).
2. **User Agent:** Will be powered by **Mistral** (via `mistralai` SDK).

Consequences

Positive	Negative
Cognitive Alignment: Gemini excels at complex, multi-step reasoning, aligning with the Minotaur's need to understand and exploit temporal dynamics and sound signatures.	**API Management Overhead:** Requires managing and monitoring the usage, billing, and credentials for **two separate LLM providers**.
Performance Efficiency: Mistral is known for high efficiency and speed when handling structured, tactical reasoning tasks, making it a cost-effective choice for the high volume of tactical decisions the User Agent makes.	**Prompt Sensitivity:** Extensive testing and prompt engineering will be required to ensure both models consistently output the exact JSON structure defined by the **Pydantic schemas** (see ADR 003).
Learning Richness: The strategic contrast between the two models provides richer failure data for the **Memory Manager** to analyze and feed back into future game runs.	

ADR 003: Communication Protocol

Positive	Negative

Field	Value
Title	Enforcement of JSON Communication via Pydantic
Status	Accepted
Date	October 4, 2025

Context

In LLM-driven games, the primary source of instability is the LLM hallucinating output formats or ignoring constraints. For this project, every action (movement, jump, grab) must be passed to and from the deterministic GSM via a reliable, parseable interface.

Decision

All inputs (`UserInput` , `MinotaurDecision`) and outputs (`GameStateResponse`) will be defined using **Pydantic models**. LangGraph will be configured to use Pydantic schemas in its tool-calling mechanism, forcing the LLMs to output valid JSON before the GSM processes the command.

Consequences

Positive	Negative
Data Integrity: Eliminates LLM hallucination from breaking the core game loop, as malformed output is caught by the Pydantic validator before reaching the GSM logic.	**Increased Development Overhead:** Requires writing verbose Pydantic schemas for every structured piece of data, adding initial boilerplate code.
Scalability to Visualization: The structured JSON output is immediately compatible with any modern front-end framework (Streamlit, React, Vue), facilitating the future **Visual MVP** phase.	
Traceability: Every turn is a structured JSON record, making debugging, logging, and iterative learning straightforward and reliable.	

Specification Document: Labyrinth: The Temporal Hunt (MVP)

This document provides the definitive specifications for the **Labyrinth: The Temporal Hunt** project. It outlines the core physics, communication protocols, and agent logic required to build the multi-agent text adventure simulator.

---

1. Core Definitions and Game Constants

All game calculations and physics are handled by the deterministic **Game State Manager (GSM)**.

Parameter	Value	Description
Grid Unit Size	meters	The base size of one *step* or *tile* in the simulation.
Maze Size (Conceptual)	km cube	Defines the scale of the world. Coordinates will be vast step integers.
User Max Height		Used for collision/LOS checks.
Minotaur Height		Emphasizes its size relative to the ceiling/floor clearance.
User Sight Radius	**2 steps (5 meters)**	Maximum distance for detecting paths or items.
Minotaur Sight Radius	**6 steps (15 meters)**	Minotaur's infrared vision advantage.
Jump Cooldown	**600 seconds (10 minutes)**	Deterministic timer for the Minotaur's temporal jump ability.
Lantern Paralysis	**120 seconds (2 minutes)**	Duration of the Minotaur's paralysis.
Lantern Respawn	**720 seconds (12 minutes)**	Cooldown after the **Lantern** is *used* before a new one can spawn.
Objectives	Red, Blue, Yellow Stones	Three unique items required to trigger the ESCAPED status.

---

2. Game State Manager (GSM) Logic

The GSM is the central authority and executes all deterministic rules based on the JSON input.

2.1 Movement and Time Flow

Command	Action	Time & Noise Calculation	Stop Conditions
MOVE [DIR] 1 (Walk/Crawl)	1 step/second. Low Noise signature. Stamina recovery.	Stops upon hitting a **COLLISION** (wall, ramp edge) or if **100 steps** are reached.	
MOVE [DIR] 2 (Run)	2 steps/second (Max speed). **High Noise** signature. Stamina depletion.	Stops upon hitting a **COLLISION**, **100 steps** reached, or **ENCOUNTER** with Minotaur.	

Command	Action	Time & Noise Calculation	Stop Conditions
GRAB / USE / LOOK / HALT	Interaction/Utility.	Consumes a minimum of **1 second** of game time.	N/A

Collision Logic: The GSM must iterate through the user's requested 100 steps. If a collision is detected at step , the user is placed at step , and the `steps_moved` is set to . The `time_taken` is calculated as .

2.2 Minotaur Chase Logic

The Minotaur's actions during the chase phase are driven by its LLM output, but constrained by the GSM's rules:

1. **Paralyzed Status:** If `lantern_active_timer > 0` , Minotaur status is **PARALYZED** regardless of other factors.
2. **Temporal Jump:** If the Minotaur LLM outputs `action: "JUMP"` , the GSM sets the status to **VANISHED** and resets the `cooldown_time` to 600s. The Minotaur's **position is held constant**.
3. **Re-entry:** If status is **VANISHED** and the simulated jump duration (5-10 seconds) has elapsed, the status is set back to **CHASING_3D** at the same positional coordinates.
4. **Encounter:** If the User and Minotaur occupy the **same grid step** while the Minotaur is **CHASING_3D** , the game ends (`status: "DEATH"`).

---

3. Communication Protocol (JSON Schemas)

All interactions are governed by **Pydantic Schemas** to ensure data integrity (N.F.R. 2.1.1).

3.1 User Input Schema (Input to GSM)

The User Agent (Mistral) or human input must conform to this structure:

JSON

```
{
  "command": "MOVE" | "HALT" | "LOOK" | "GRAB" | "USE",
  "direction": "NORTH" | "SOUTH" | "UP RAMP" | null,
  "steps": 100,
  "speed": 1 | 2,
  "target": "RED STONE" | "LANTERN" | null
}
```

3.2 Minotaur Decision Schema (Output from Gemini)

The Minotaur Agent must return its calculated action based on this structure:

JSON

```
{
  "action": "PATHFIND" | "JUMP" | "WAIT" | "CHASE",
  "target_coords": {"x": 0, "y": 0, "z": 0} | null
}
```

3.3 Game State Response Schema (Output from GSM)

The GSM's primary output to the Streamlit UI and the next agent turn:

JSON

```
{
  "status": "SUCCESS" | "DEATH" | "ESCAPED" | "ERROR",
  "user_state": {
    "position": {"x": 0, "y": 0, "z": 0},
    "stamina_pct": 0.85,
    "inventory": ["RED STONE", "LANTERN"],
    "lantern_cooldown": 720
  },
  "environment": {
    "visible_paths": ["NORTH", "DOWN RAMP"],
    "visible_items": ["BLUE STONE"],
    "message": "Current Z-Level: 3",
    "steps_moved": 10,
    "time_taken": 5,
    "stop_reason": "COLLISION" | "ENCOUNTER" | "SUCCESS"
  },
  "minotaur_cue": {
    "proximity": "VERY CLOSE" | "VANISHED" | "PARALYZED",
    "audio_direction": "EAST" | null,
    "temporal_status": "CHASING_3D",
    "cooldown_time": 450
  },
  "raw_text_output": "The Minotaur materializes at its fixed re-entry position!"
}
```

---

4. Agent Logic and Learning (N.F.R. 3.2.2)

The LLM agents must read from and contribute to the persistent memory system to adapt their strategies over successive runs.

4.1 Minotaur Agent (Gemini) - Learning Objectives

The Gemini LLM's **System Prompt** will be injected with a summary of past **User Escapes (Minotaur Losses)**.

- **Primary Focus:** Learning the Mistral Agent's **predictive evasion patterns** (e.g., "User always runs West after hearing the Temporal Whine").
- **Action Optimization:** Determining the optimal conditions (proximity, line of sight, remaining cooldown) under which to execute the rare **Temporal Jump**.

4.2 User Agent (Mistral) - Learning Objectives

The Mistral LLM's **System Prompt** will be injected with a summary of past **User Deaths (Mistral Losses)**.

- **Primary Focus: Risk/Reward Calculation** regarding noise and speed. Learning the coordinates of critical resources (Ramps, Stones, Lanterns).
- **Resource Timing:** Learning to time the use of the **Lantern** precisely to cover the grab of an exposed Stone or escape a near-death situation.

4.3 Memory Injection Flow

1. **Setup Phase:** LangGraph calls the **Memory Manager** to retrieve structured reflections summarizing the last 5 relevant game runs.
2. **Prompt Construction:** The retrieved reflections are formatted into a concise **{learning_injection}** string (e.g., "STRATEGIC REVISION: Avoid Z=5, Minotaur Jump effectiveness is high there.")
3. **Execution:** The complete prompt (Persona + Constraints + Learning Injection) is passed to the respective LLM for decision-making.

Specification Document: Labyrinth: The Temporal Hunt (MVP)

This document provides the definitive specifications for the **Labyrinth: The Temporal Hunt** project. It outlines the core physics, communication protocols, and agent logic required to build the multi-agent text adventure simulator.

---

1. Core Definitions and Game Constants

All game calculations and physics are handled by the deterministic **Game State Manager (GSM)**.

Parameter	Value	Description
Grid Unit Size	meters	The base size of one *step* or *tile* in the simulation.
Maze Size (Conceptual)	km cube	Defines the scale of the world. Coordinates will be vast step integers.
User Max Height		Used for collision/LOS checks.
Minotaur Height		Emphasizes its size relative to the ceiling/floor clearance.
User Sight Radius	**2 steps (5 meters)**	Maximum distance for detecting paths or items.
Minotaur Sight Radius	**6 steps (15 meters)**	Minotaur's infrared vision advantage.
Jump Cooldown	**600 seconds (10 minutes)**	Deterministic timer for the Minotaur's temporal jump ability.
Lantern Paralysis	**120 seconds (2 minutes)**	Duration of the Minotaur's paralysis.
Lantern Respawn	**720 seconds (12 minutes)**	Cooldown after the **Lantern** is *used* before a new one can spawn.
Objectives	Red, Blue, Yellow Stones	Three unique items required to trigger the ESCAPED status.

---

2. Game State Manager (GSM) Logic

The GSM is the central authority and executes all deterministic rules based on the JSON input.

2.1 Movement and Time Flow

Command	Action	Time & Noise Calculation	Stop Conditions
MOVE [DIR] 1 **(Walk/Crawl)**	1 step/second. Low Noise signature. Stamina recovery.	Stops upon hitting a **COLLISION** (wall, ramp edge) or if **100 steps** are reached.	
MOVE [DIR] 2 **(Run)**	2 steps/second (Max speed). **High Noise** signature. Stamina depletion.	Stops upon hitting a **COLLISION**, **100 steps** reached, or **ENCOUNTER** with Minotaur.	
GRAB / USE / LOOK / HALT	Interaction/Utility.	Consumes a minimum of **1 second** of game time.	N/A

Collision Logic: The GSM must iterate through the user's requested 100 steps. If a collision is detected at step , the user is placed at step , and the `steps_moved` is set to . The `time_taken` is calculated as .

2.2 Minotaur Chase Logic

The Minotaur's actions during the chase phase are driven by its LLM output, but constrained by the GSM's rules:

1. **Paralyzed Status:** If `lantern_active_timer > 0` , Minotaur status is **PARALYZED** regardless of other factors.
2. **Temporal Jump:** If the Minotaur LLM outputs `action: "JUMP"` , the GSM sets the status to **VANISHED** and resets the `cooldown_time` to 600s. The Minotaur's **position is held constant**.
3. **Re-entry:** If status is **VANISHED** and the simulated jump duration (5-10 seconds) has elapsed, the status is set back to **CHASING_3D** at the same positional coordinates.
4. **Encounter:** If the User and Minotaur occupy the **same grid step** while the Minotaur is **CHASING_3D** , the game ends (`status: "DEATH"`).

---

3. Communication Protocol (JSON Schemas)

All interactions are governed by **Pydantic Schemas** to ensure data integrity (N.F.R. 2.1.1).

3.1 User Input Schema (Input to GSM)

The User Agent (Mistral) or human input must conform to this structure:

JSON

```
{
  "command": "MOVE" | "HALT" | "LOOK" | "GRAB" | "USE",
  "direction": "NORTH" | "SOUTH" | "UP RAMP" | null,
  "steps": 100,
  "speed": 1 | 2,
  "target": "RED STONE" | "LANTERN" | null
}
```

3.2 Minotaur Decision Schema (Output from Gemini)

The Minotaur Agent must return its calculated action based on this structure:

JSON

```
{
  "action": "PATHFIND" | "JUMP" | "WAIT" | "CHASE",
  "target_coords": {"x": 0, "y": 0, "z": 0} | null
}
```

3.3 Game State Response Schema (Output from GSM)

The GSM's primary output to the Streamlit UI and the next agent turn:

JSON

```
{
  "status": "SUCCESS" | "DEATH" | "ESCAPED" | "ERROR",
  "user_state": {
    "position": {"x": 0, "y": 0, "z": 0},
    "stamina_pct": 0.85,
    "inventory": ["RED STONE", "LANTERN"],
    "lantern_cooldown": 720
  },
  "environment": {
    "visible_paths": ["NORTH", "DOWN RAMP"],
    "visible_items": ["BLUE STONE"],
    "message": "Current Z-Level: 3",
    "steps_moved": 10,
    "time_taken": 5,
    "stop_reason": "COLLISION" | "ENCOUNTER" | "SUCCESS"
  },
  "minotaur_cue": {
    "proximity": "VERY CLOSE" | "VANISHED" | "PARALYZED",
    "audio_direction": "EAST" | null,
    "temporal_status": "CHASING_3D",
    "cooldown_time": 450
  },
  "raw_text_output": "The Minotaur materializes at its fixed re-entry position!"
}
```

---

4. Agent Logic and Learning (N.F.R. 3.2.2)

The LLM agents must read from and contribute to the persistent memory system to adapt their strategies over successive runs.

4.1 Minotaur Agent (Gemini) - Learning Objectives

The Gemini LLM's **System Prompt** will be injected with a summary of past **User Escapes (Minotaur Losses)**.

- **Primary Focus:** Learning the Mistral Agent's **predictive evasion patterns** (e.g., "User always runs West after hearing the Temporal Whine").
- **Action Optimization:** Determining the optimal conditions (proximity, line of sight, remaining cooldown) under which to execute the rare **Temporal Jump**.

4.2 User Agent (Mistral) - Learning Objectives

The Mistral LLM's **System Prompt** will be injected with a summary of past **User Deaths (Mistral Losses)**.

- **Primary Focus: Risk/Reward Calculation** regarding noise and speed. Learning the coordinates of critical resources (Ramps, Stones, Lanterns).
- **Resource Timing:** Learning to time the use of the **Lantern** precisely to cover the grab of an exposed Stone or escape a near-death situation.

4.3 Memory Injection Flow

1. **Setup Phase:** LangGraph calls the **Memory Manager** to retrieve structured reflections summarizing the last 5 relevant game runs.
2. **Prompt Construction:** The retrieved reflections are formatted into a concise `{learning_injection}` string (e.g., "STRATEGIC REVISION: Avoid Z=5, Minotaur Jump effectiveness is high there.")
3. **Execution:** The complete prompt (Persona + Constraints + Learning Injection) is passed to the respective LLM for decision-making.

This Guideline document establishes the necessary coding and architectural practices for developing **Labyrinth: The Temporal Hunt**, ensuring the integrity of the deterministic game logic and the effective integration of the AI agents.

Implementation Guidelines: Labyrinth: The Temporal Hunt

1. Foundational Coding and Environment

These standards govern the setup and execution environment, ensuring reproducibility and reliability.

Guideline	Description	Rationale
Purity of Environment	The project **must** be built and run inside the **Docker** container defined by `Dockerfile` and `requirements.txt`.	Guarantees all developers/agents use the exact same versions of Streamlit, LangGraph, Pydantic, Gemini, and Mistral SDKs.
Configuration	API keys and sensitive data **must** be loaded from the local `.env` file using `python-dotenv`. **Hardcoding keys is forbidden.**	Security and portability.
Error Handling	All functions involving external network calls (i.e., the LLM API calls) **must** use robust `try...except` blocks to handle connection failures, timeouts, and API errors.	Prevents the entire application from crashing due to transient network issues.

---

2. The Golden Rule: JSON Protocol Enforcement

This is the non-negotiable rule governing the interaction between the deterministic GSM and the cognitive LLM agents.

2.1 Pydantic Validation

- **Input Gate:** All data flowing **into** the GSM (the User Agent's `UserInput` and the Minotaur's `MinotaurDecision`) **MUST** be validated using the Pydantic schemas. The GSM should only proceed if `model_validate_json()` is successful.
- **Output Consistency:** The GSM's final output (`GameStateResponse`) **MUST** strictly conform to its Pydantic schema before being sent to the Streamlit UI or the next agent (N.F.R. 2.1.1).

2.2 LLM Output Handling

- **Forced JSON:** The **System Prompts** for both Gemini and Mistral must be engineered to explicitly request a JSON output that matches the defined schema.
- **Failure Mode:** If an LLM returns a non-conforming JSON or an empty string, the system **must not crash**. Instead:
  - **Minotaur:** Must default to the safest action (e.g., `action: "WAIT"`) and log the validation error.
  - **User Agent:** Must default to `command: "LOOK"` to reassess the situation.

---

3. GSM Implementation Guidelines (Deterministic Logic)

The `GameStateManager` is responsible for all non-cognitive rules.

Area	Guideline	Constraints & Rules
Separation of Concerns	The GSM class **MUST NOT** contain any direct calls to LLM APIs or access agent-specific logic. It is purely a physics engine (ADR 001).	Integrity of deterministic rules.
Time Advancement	The calculated `time_taken` (F.R. 1.2.3) from the user's movement **must** be the universal decrement value for all timers (`minotaur_jump_timer`, `lantern_respawn_timer`).	Ensures consistent physics across every turn.
Movement & Collision	The `_simulate_movement` method must be written to respect the `speed` parameter and calculate the precise step number where a **COLLISION** occurs, returning `steps_moved` and `time_taken`.	Adherence to F.R. 1.1.5 (Collision Logic).
Temporal Logic	The Minotaur's jump and re-entry logic must strictly enforce **F.R. 2.1.1 (Positional Jump)**: The coordinates remain identical during the `VANISHED` state.	Critical to the core 4D game mechanic.
Resource Logic	The Lantern respawn timer logic must enforce the **12-minute (720s) cooldown** after paralysis ends, and track the single instance of the **Lantern** in the `visible_items` list.	Adherence to F.R. 2.2.2 and F.R. 2.2.3.

---

4. Agent Integration Guidelines (LLM Logic)

These guidelines govern the cognitive part of the system, orchestrated by LangGraph.

Agent	LLM Model	Strategic Prompt Focus
Minotaur	**Gemini** (ADR 002)	**Predictive Pursuit & Ambush Timing:** Prompt must heavily rely on `audio_direction` and `cooldown_time` to decide between `CHASE` and the `JUMP` action. Must use its **6-step sight advantage**.
User	**Mistral** (ADR 002)	**Risk-Averse Mapping & Resource Management:** Prompt must prioritize **stealth** (`speed: 1`) when `proximity` is above `FAINT`. Must factor `stamina_pct` into its decision to use `RUN`.
Prompt Delivery	Both LLMs	The full system prompt must be constructed to include the strict constraints, the LLM's persona, **AND** the dynamic learning injection from the memory system.

4.1 Learning System Integration (Memory Manager)

- **Injection Point:** The `memory_manager.get_latest_reflections()` function must be called **only during the game setup/initialization phase** (before the first turn) to retrieve the relevant strategic guidance.

- **Reflection Logic:** The Minotaur must learn from past **User Escapes (Losses)**, and the User must learn from past **User Deaths (Losses)** (N.F.R. 3.2.2). This selective retrieval ensures focused strategic adaptation.

---

5. UI Visualization Guidelines (Streamlit)

- **Data Source:** The UI **must only read** the `GameStateResponse` JSON object. It should never access the raw internal state of the GSM.
- **Proximity Alert:** Use color coding and appropriate UI elements (e.g., `st.warning`, `st.error`) to visualize the `minotaur_cue.proximity` (e.g., `VERY CLOSE` = Red; `FAINT` = Yellow).
- **Input Interface:** The input text box must clearly guide the user/testing agent with examples of the required command syntax (`MOVE NORTH RUN`, `GRAB STONE`).
- **Log Display:** The main log panel should display both the **`raw_text_output`** (for human readability) and the summary status metrics (`time_taken`, `steps_moved`) for analytical clarity.This Guideline document establishes the necessary coding and architectural practices for developing **Labyrinth: The Temporal Hunt**, ensuring the integrity of the deterministic game logic and the effective integration of the AI agents.

Implementation Guidelines: Labyrinth: The Temporal Hunt

1. Foundational Coding and Environment

These standards govern the setup and execution environment, ensuring reproducibility and reliability.

Guideline	Description	Rationale
Purity of Environment	The project **must** be built and run inside the **Docker** container defined by `Dockerfile` and `requirements.txt`.	Guarantees all developers/agents use the exact same versions of Streamlit, LangGraph, Pydantic, Gemini, and Mistral SDKs.
Configuration	API keys and sensitive data **must** be loaded from the local `.env` file using `python-dotenv`. **Hardcoding keys is forbidden.**	Security and portability.
Error Handling	All functions involving external network calls (i.e., the LLM API calls) **must** use robust `try...except` blocks to handle connection failures, timeouts, and API errors.	Prevents the entire application from crashing due to transient network issues.

---

2. The Golden Rule: JSON Protocol Enforcement

This is the non-negotiable rule governing the interaction between the deterministic GSM and the cognitive LLM agents.

2.1 Pydantic Validation

- **Input Gate:** All data flowing **into** the GSM (the User Agent's `UserInput` and the Minotaur's `MinotaurDecision`) **MUST** be validated using the Pydantic schemas. The GSM should only proceed if `model_validate_json()` is successful.
- **Output Consistency:** The GSM's final output (`GameStateResponse`) **MUST** strictly conform to its Pydantic schema before being sent to the Streamlit UI or the next agent (N.F.R. 2.1.1).

2.2 LLM Output Handling

- **Forced JSON:** The **System Prompts** for both Gemini and Mistral must be engineered to explicitly request a JSON output that matches the defined schema.
- **Failure Mode:** If an LLM returns a non-conforming JSON or an empty string, the system **must not crash**. Instead:
  - **Minotaur:** Must default to the safest action (e.g., `action: "WAIT"`) and log the validation error.
  - **User Agent:** Must default to `command: "LOOK"` to reassess the situation.

---

3. GSM Implementation Guidelines (Deterministic Logic)

The `GameStateManager` is responsible for all non-cognitive rules.

Area	Guideline	Constraints & Rules
Separation of Concerns	The GSM class **MUST NOT** contain any direct calls to LLM APIs or access agent-specific logic. It is purely a physics engine (ADR 001).	Integrity of deterministic rules.
Time Advancement	The calculated `time_taken` (F.R. 1.2.3) from the user's movement **must** be the universal decrement value for all timers (`minotaur_jump_timer`, `lantern_respawn_timer`).	Ensures consistent physics across every turn.
Movement & Collision	The `simulate_movement` method must be written to respect the `speed` parameter and calculate the precise step number where a **COLLISION** occurs, returning `steps_moved` and `time_taken`.	Adherence to F.R. 1.1.5 (Collision Logic).
Temporal Logic	The Minotaur's jump and re-entry logic must strictly enforce **F.R. 2.1.1 (Positional Jump)**: The coordinates remain identical during the `VANISHED` state.	Critical to the core 4D game mechanic.
Resource Logic	The Lantern respawn timer logic must enforce the **12-minute (720s) cooldown** after paralysis ends, and track the single instance of the **Lantern** in the `visible_items` list.	Adherence to F.R. 2.2.2 and F.R. 2.2.3.

---

4. Agent Integration Guidelines (LLM Logic)

These guidelines govern the cognitive part of the system, orchestrated by LangGraph.

Agent	LLM Model	Strategic Prompt Focus
Minotaur	**Gemini** (ADR 002)	**Predictive Pursuit & Ambush Timing:** Prompt must heavily rely on `audio_direction` and `cooldown_time` to decide between `CHASE` and the `JUMP` action. Must use its **6-step sight advantage**.
User	**Mistral** (ADR 002)	**Risk-Averse Mapping & Resource Management:** Prompt must prioritize **stealth** (`speed: 1`) when `proximity` is above `FAINT`. Must factor `stamina_pct` into its decision to use `RUN`.
Prompt Delivery	Both LLMs	The full system prompt must be constructed to include the strict constraints, the LLM's persona, **AND** the dynamic learning injection from the memory system.

4.1 Learning System Integration (Memory Manager)

- **Injection Point:** The `memory_manager.get_latest_reflections()` function must be called **only during the game setup/initialization phase** (before the first turn) to retrieve the relevant strategic guidance.
- **Reflection Logic:** The Minotaur must learn from past **User Escapes (Losses)**, and the User must learn from past **User Deaths (Losses)** (N.F.R. 3.2.2). This selective retrieval ensures focused strategic adaptation.

---

5. UI Visualization Guidelines (Streamlit)

- **Data Source:** The UI **must only read** the `GameStateResponse` JSON object. It should never access the raw internal state of the GSM.
- **Proximity Alert:** Use color coding and appropriate UI elements (e.g., `st.warning`, `st.error`) to visualize the `minotaur_cue.proximity` (e.g., `VERY CLOSE` = Red; `FAINT` = Yellow).
- **Input Interface:** The input text box must clearly guide the user/testing agent with examples of the required command syntax (`MOVE NORTH RUN`, `GRAB STONE`).
- **Log Display:** The main log panel should display both the **`raw_text_output`** (for human readability) and the summary status metrics (`time_taken`, `steps_moved`) for analytical clarity.
