Dark Pages – Horror Game Demo Build

Logan Rusch & Steven Taranto
Department of Computer Science, University of New Hampshire, Durham, NH 03824

Introduction

Context
A first-person stealth horror demo spanning one main level, “Floor 5”.

Project Vision
Create a suspenseful, replayable horror experience that validates immersive stealth mechanics and intelligent antagonist behavior.

Value & Success Criteria
Deliver fully functional AI with unique behavioral patterns, polished core player mechanics (movement, interaction, lighting), and integrated telemetry to generate actionable playtest feedback.

Requirements

AI Architecture
Prioritizes modularity and scalability. AI logic is separated from reusable stealth utilities, keeping blueprint communication channels limited and efficient.

Telemetry Flow
Data routes from gameplay events directly to external dashboards for evaluation, providing low-cost, high-yield behavioral insights.

Technical Rationale
Upgraded to Unreal Engine 5.6 to utilize GameAnalytics for its comprehensive free-tier dashboards, replacing built-in analytics.

Project Design

Functional Requirements
- AI responds dynamically to player visibility.
- Players must navigate, interact with objects, and use a lighting system.
- Audio, environmental lighting, and UI objectives adapt to the player’s state.

Non-Functional Requirements
- Optimized for lower-end systems: Intel i5, UHD 620 graphics, 8GB RAM.
- 720p at 25–30 FPS target; level load times under 10 seconds; build size under 5GB.

Gameplay

First Person View Of The AI Enemy.

Image #1 A screenshot of gameplay showing the AI that was programmed

AI Diagrams

AI-Hide Pseudocode

```
FUNCTION FindCoverPoint(Player, HidingObject, World, CoverOffset):

  IF Player is null OR HidingObject is null OR World is null:
     RETURN a default vector (e.g., zero vector)

  // 1. Get the bounding box of the hiding object
  GET center = HidingObject.Bounds.Center
  GET extent = HidingObject.Bounds.Extent   // half-size of the object

  // 2. Compute the direction from the hiding object to the player
  directionToPlayer = Normalize(Player.Position - center)

  // 3. Flip the direction so it's pointing AWAY from the player
  directionAwayFromPlayer = -directionToPlayer

  // 4. Compute the hiding spot behind the object
  //    Distance = object size + optional offset
  hidingSpot = center + directionAwayFromPlayer * (extent.Size + CoverOffset)

  // 5. Check if line of sight from player to hiding spot is blocked
  isBlocked = LineTrace(
              start = Player.Position,
              end   = hidingSpot,
              channel = Visibility )

  // 6. Return the chosen hiding location
  RETURN hidingSpot
END FUNCTION
```

Image #2 Pseudocode of an algorithm for calculating correct hiding spots for the AI to move to.

UML Diagram of the AI Enemy

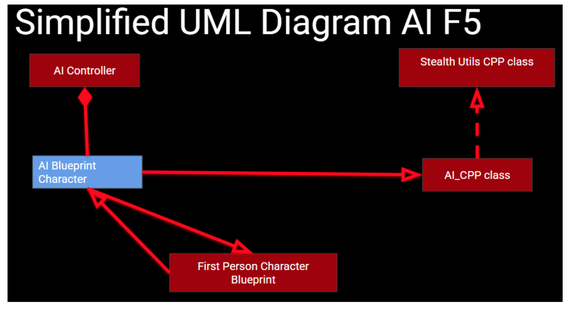

Image #3 A UML diagram showing how the enemy AI’s behavior is decoupled into different classes.

Telemetry Diagrams

Data Flow Diagram of Telemetry

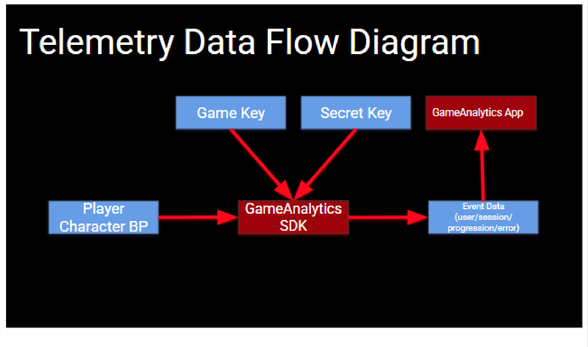

Image #4 An image showing how data flows from different services and classes for telemetry to work.

Dashboard of Telemetry Back End

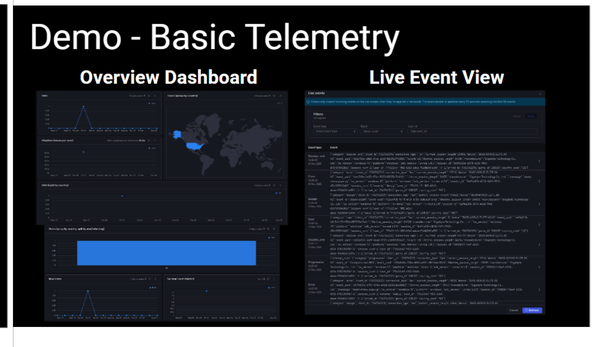

Image #5 Screenshots of the data from the demo game being displayed on the back end of the telemetry.

Implementation / Testing

AI Implementation (Floor 5)
Built using UE 5.6 Blueprints and a custom C++ stealth utility class. The system continuously calculates the player’s forward vector and field of view. If the AI falls within this cone, an unobstructed line trace confirms visibility and triggers the enemy’s behavioral response.

AI Testing
Relied on targeted debug print statements and extensive manual playtesting to evaluate qualitative factors like pacing, tension, and game feel.

Telemetry Implementation
Integrated the GameAnalytics SDK via the UE Marketplace. A UGameAnalytics instance in the character blueprint initializes sessions and tracks progression, player deaths, and system errors.

Telemetry Testing
Validated data tracking accuracy through the GameAnalytics live events stream and Unreal Engine diagnostic logs.

Evaluation and Conclusion

Evaluation
The demo successfully demonstrates the core horror loop. The AI accurately detects and reacts to the player, and telemetry reporting proves stable and effective at capturing required metrics.

Limitations
UI, saving/loading systems, and final art assets require further polish before full deployment.

Next Steps
Optimize performance for stability on target hardware specifications, finalize UI elements, and conduct structured closed playtests with 5-10 users to gather actionable feedback.

Acknowledgements

Project Sponsor: Maxim Budyansky

Advisor: Lisa Henry

Abstract

Dark Pages is a first-person horror game demo designed to evaluate stealth-based gameplay, intelligent enemy behavior, and player experience through integrated telemetry systems. The project focuses on modular AI design using Unreal Engine, reusable stealth utilities, and analytics-driven validation of core gameplay loops. Development emphasizes the primary antagonist AI for Floor 5, utilizing field-of-view checks and line traces to trigger dynamic responses via a scalable C++ stealth class. Simultaneously, telemetry captures session data, progression, and error tracking to inform iterative design. The resulting demo successfully serves as a technical proof-of-concept, validating the core tension mechanics while generating actionable player feedback to guide future development and stakeholder decision-making.