Documentación Técnica: Prototipo de Sigilo con Pathfinding

1. Fundamento Tecnológico: NavMesh (Malla de Navegación)

Para la resolución de caminos, el proyecto utiliza el sistema **Unity Navigation Mesh**.

- **Algoritmo Base:** Unity implementa internamente una variación del algoritmo **A* (A-Star)**. Este algoritmo busca el camino de menor coste entre dos nodos en un grafo.
- **Implementación:** Se utilizó el componente NavMeshSurface para "hornear" (bake) la geometría estática de la escena (suelos y obstáculos), creando una representación abstracta de las superficies transitables (color azul en el editor). Esto permite que los agentes eviten obstáculos automáticamente sin programación manual de colisiones complejas.

2. Mecánica 1: Movimiento del Jugador (Point & Click)

Descripción

El jugador controla al personaje haciendo clic en cualquier punto del escenario, y el personaje busca la ruta óptima para llegar allí.

Algoritmo y Lógica

Se utiliza una técnica conocida como **Raycasting** (Lanzamiento de Rayos) combinada con el sistema de entrada.

1. **Detección (Raycast):** Al hacer clic, se proyecta un rayo invisible desde la cámara (espacio 2D de la pantalla) hacia el mundo 3D.
2. **Resolución de Camino:** Si el rayo impacta con el NavMesh (el suelo transitable), se extraen las coordenadas del punto de impacto (hit.point).
3. **Pathfinding:** Se invoca el método agent.SetDestination(hit.point). En este momento, el NavMeshAgent calcula la ruta A* evitando obstáculos estáticos y aplica la velocidad y aceleración definidas para mover al personaje frame a frame.

Snippet de Integración:

C#

// Convierte la posición del mouse en un rayo 3D

Ray ray = Camera.main.ScreenPointToRay(mousePos);

// Si golpea el NavMesh, asigna el destino al agente

if (Physics.Raycast(ray, out hit)) { agent.SetDestination(hit.point); }

3. Mecánica 2: IA Enemiga (Sistema de Waypoints)

Descripción

El enemigo patrulla de forma autónoma entre una serie de puntos predefinidos (GameObjects vacíos) en un bucle infinito.

Algoritmo y Lógica

Se implementa una **Máquina de Estados Finitos (FSM)** muy simplificada basada en una lista secuencial de objetivos.

1. **Almacenamiento:** Los puntos de ruta se almacenan en un arreglo (Transform[] waypoints).
2. **Evaluación de Distancia:** En cada frame (Update), el script consulta al NavMeshAgent mediante agent.remainingDistance.
3. **Transición:** Si la distancia restante es menor a un umbral (ej. 0.5 metros) y el agente no está calculando una ruta (!agent.pathPending), se considera que ha llegado.
4. **Selección Modular:** El siguiente punto se selecciona utilizando la operación aritmética **módulo** (%). Esto permite que, al llegar al último punto del array, el índice vuelva automáticamente a 0, creando un ciclo infinito sin necesidad de condicionales complejos.

Fórmula de selección:

$$ÍndiceSiguiente = (ÍndiceActual + 1) \pmod{TotalPuntos}$$

4. Control de Cámara (Observación)

Descripción

Permite al jugador rotar la vista horizontalmente para observar el entorno y localizar al enemigo sin mover al personaje.

Lógica

Se utiliza la transformación directa de los **Ángulos de Euler**. Al presionar las teclas A o D (detectadas por el *New Input System*), se aplica una rotación sobre el eje Y (Vector Up) del mundo. Esto simula el movimiento de "cuello" del observador.